Summary
2014-04-01, 15:04:28

Differences exist between documents.

New Document:	Old Document:
draft_10	draft_9
148 pages (9.71 MB)	134 pages (9.69 MB)
2014-04-01, 15:03:10	2014-04-01, 15:03:06
Used to display results.	

Get started: first change is on page 1.

No pages were deleted

How to read this report

Highlight indicates a change.
~~Deleted~~ indicates deleted content.
 indicates pages were changed.
 indicates pages were moved.

Designing The EHR

Martin Wehlou

April 1, 2014

MAN IN THE MIDDLE BOOKS

First published in book form June 2014.
Cover design by Gilles Vandenoostende
Printed edition ISBN 978-91-981706-0-3

Composed in Scrivener®
Formatted for LaTeX
by MultiMarkdown

Contents

Contents iii

Draft versions 1

Introduction 3

Terminology 7

Acknowledgements 11

1 About the author 13

I The basics 23

2 Why are we doing this? 27
2.1 Making people healthier . . . 27
2.2 Quicker diagnosis and treatment . . . 28
2.3 Simpler . . . 28
2.4 Safer . . . 28
2.5 Analysis of what we're doing . . . 29
2.6 Economic tracking . . . 29
2.7 More transparent to the patient . . . 29
2.8 Reduce accidents . . . 30
2.9 Increase consistency . . . 30
2.10 Reducing need for doctors . . . 31
2.11 Playing with computers . . . 31
2.12 Agree on the purpose . . . 31

3 The business model 33
3.1 Large scale business model . . . 33
3.2 Small scale business model . . . 34

4 What are doctors made of? 37
4.1 Theory of the healthy human . . . 37
4.2 Mechanisms of disease . . . 38
4.3 Clinical examinations . . . 38
4.4 Craftsmanship . . . 39
4.5 Diagnostic and therapeutic knowledge . . . 40

5 The History of medical records 41
5.1 The absence of records . . . 41
5.2 Paperbased mementos . . . 42

6 The stakeholders 45

7 How should the EHR assist us? 47
7.1 Compare to other knowledge areas . . . 47
7.2 What should I do? . . . 49
7.3 How should I do it? . . . 50
7.4 What did I forget? . . . 51
7.5 History in context . . . 51

II Current systems 53

8 The goal of the system 57

9 Current EHR: Cosmic 59

10 Knowledge support 63
10.1 Original articles . . . 63
10.2 Review articles . . . 64
10.3 Textbooks . . . 64
10.4 Guidelines . . . 64
10.5 Continued Professional Education . . . 70

11 How is the record created? 73
11.1 The input method . . . 73
11.2 The different results . . . 75
11.3 How I am doing it . . . 76
11.4 Where is the record created? . . . 77

12 The information model 79
12.1 Improvements . . . 84

III Obstacles and bad ideas 87

13 Research the EHR (fallacy) 91
13.1 Anecdotal medicine . . . 91
13.2 Evidence based medicine . . . 92
13.3 Electronic astrology . . . 94
13.4 Exceptions . . . 94

14 How management data corrupts 97

15 Beware of false requirements 99
15.1 One patient - one medical record . . . 101
15.2 A common database . . . 102
15.3 Terminology systems . . . 104

16 Avoid the need for standards 105
16.1 Lab codes . . . 105
16.2 Terminologies . . . 106
16.3 What is the benefit of common terms? . . . 108

17 The oversimplification of processes 113
17.1 My amended version . . . 115
17.2 Labeling the disaster . . . 117

18 How to connect 119
18.1 Big fat and ugly . . . 119
18.2 Inter-app messaging . . . 120
18.3 Inter-database messaging . . . 121
18.4 Basic interfacing principles . . . 123

19 What is the difference? 129
19.1 Different kinds of data . . . 129
19.2 Indeterminate communication partners . . . 130
19.3 Eternal data . . . 130
19.4 Write-only data . . . 131

20 The perils of user groups 133

21 The politics of purchasing 137
21.1 The purchasing organisation . . . 137
21.2 The manager's viewpoint . . . 138
21.3 The user's viewpoint . . . 139
21.4 The user group . . . 139
21.5 The support organization . . . 141

21.6 When the shit hits the fan . . . 143
21.7 So, how do you fix this? . . . 144

22 OpenEHR **147**

IV A consistent design **149**

23 The phases of the clinical process **153**
23.1 Clinical encounter . . . 153
23.2 Overview of patient history . . . 154
23.3 Clinical examination . . . 157
23.4 Creating referrals and orders . . . 160
23.5 Creating prescriptions . . . 162
23.6 Creating the note record . . . 164
23.7 Finding results . . . 166
23.8 Receiving results . . . 167
23.9 Reporting . . . 169
23.10 Reporting to national registries . . . 171
23.11 The real requirements . . . 172

24 Encapsulation **181**

25 How active should the software be? **185**
25.1 The keyhole effect . . . 185
25.2 The indiscriminate criteria effect . . . 185
25.3 The disempowerment effect . . . 186
25.4 Nurse vs doctor domain expert . . . 186

26 Document tree **189**
26.1 The attention list . . . 193
26.2 Encryption . . . 199

27 The size of the system **201**
27.1 The political aspect . . . 202
27.2 The development aspect . . . 202
27.3 The flexibility aspect . . . 203
27.4 Document tree aspect . . . 203

28 Rules and audits **205**

29 General and special **207**
29.1 Process oriented . . . 207
29.2 Document oriented . . . 208

30 Interconnections **209**

31 Presentation elements **211**
31.1 The timeline visualization 211
31.2 The anatomical visualization 212
31.3 Word clouds or tags 212
31.4 Issue oriented 213

32 Coding systems **217**
32.1 Multiple coding systems 217
32.2 Updating and changes 218
32.3 Finding code tables 218

33 Touch or not **219**

34 Security **221**
34.1 Confidentiality 222
34.2 Integrity 230
34.3 Availability 230

Appendices **235**

Attacking hashed personal numbers **235**

Registry anonymizing **237**

Document tree design **251**

Draft versions

Table 0.1: Draft versions

Version	Date	Remarks
1	January 23, 2014	
2	February 2, 2014	
3	February 6, 2014	
4	February 12, 2014	
5	February 16, 2014	
6	February 24, 2014	
7	March 23, 2014	
8	March 24, 2014	
9	March 25, 2014	Started the "draft Tuesday" release schedule
10	April 1, 2014	

Download links

From version 9 and onwards, I'll post preview versions every Tuesday to the following URL:

`http://wehlou.com/ehrbook/draft.pdf`

Older versions will be preserved as:

`http://wehlou.com/ehrbook/draft_X.pdf`

... where "X" is a one or two digit number. For instance:

`http://wehlou.com/ehrbook/draft_9.pdf`

and

`http://wehlou.com/ehrbook/draft_10.pdf`

. . . and so on. The first one in the series is number 9. At times, I may also post a diff file, that is a PDF marked with the differences since the previous draft. For the most recent draft, the diff will be called:

`http://wehlou.com/ehrbook/diff.pdf`

while for older releases they'll look like:

`http://wehlou.com/ehrbook/diff_9.pdf`

In this last example, the diff consists of the version 9 draft, with the differences marked as compared to version 8. I'll only start doing diffs from version 10.

Contributing comments

You'll make me absolutely ecstatic if you contribute with comments of all kinds, such as errata, arguments for or against what I'm saying, or suggestions for expansion or other reading.

You can do that by any means that suits you, I'll find a way to use it. So go ahead and write me emails, send me scribbled notes on birch tree bark, a mix tape, video confessions, or whatever, but one way that's easy both for you and for me is annotated PDF files. You can annotate my PDF using any of these suggested tools:

- Acrobat Pro.
- Preview on OSX (annotation bar).
- GoodReader for iOS.

. . . or any number of other tools, some of which must exist for Windows. If you want, I could prepare a special "for comment" version PDF that you can annotate with just Acrobat Reader. Let me know by email, so I know it's worth doing.

You can send the annotated file back to me at:

martin@wehlou.com

If you *don't* want a mention in my "Acknowledgements", please say so. Else I'll include you.

Thanks!

Introduction

These notes started out as a brief outline for my yearly lecture at the Karolinska Institute in Stockholm, to the students of the "International Masters Program in Health Informatics". As time went by, I began to suspect that these notes harbored in them something bigger, maybe even a book. After some procrastination, I finally started expanding on these notes in January 2014, and what you see here is the result.

The target audience for the original notes consisted of students well-versed in both medicine and technology, which explains why the notes tend to slide from one area into the other without much of a transition.

Converting these relatively limited notes to a full book implies that the target audience also changes in composition and character. Except for the aforementioned students, I include doctors, nurses, and software engineers into the intended audience. I think it absolutely necessary that software engineers learn to appreciate the nuances of doctors' use of the medical record, while I also think it essential that doctors learn to understand the technical limitations and possibilities inherent in these systems. Real solutions will not come from two or more professional groups working together, but will only come from each professional group reaching into the other's knowledge area and grabbing onto the stuff that they actually need. The design coherence needed for fully useful system designs must sprout from minds that can bridge the gap, and if these are in short supply, we must either produce more of them, utilize them better, or both.

The technical audience should be able to read the entire book, without skipping anything.

Doctors interested in helping produce better tools are also an intended audience, but I do realize that some chapters and appendices may be a bit beyond this audience, and not so useful. So if you're a doctor, you're allowed to skip over stuff that seems a bit too boring and technical.

Another very important point is this: don't look back! We have built IT systems based on how paper records work, and that didn't turn out well. We also have to stop looking at current systems for inspiration on how

to build the next generation system, else they'll also be dismal failures[1]. Forget about the past. Think up new systems from first principles, and it wouldn't even hurt if you assumed that whatever looks most like what we have today is a bad idea and should be scrapped. The more different from that, the better.

There are also a number of evolutions in the development of the EHR that worry me. The structuring of the *current* form of the medical records is often done to "make the computer understand" what is happening in the healthcare process. If we think about the interaction of man and machine in healthcare, there are three ways to go:

- Man does his job and then feeds the machine data for safekeeping and management analysis purposes. This is what we have today for the main part.

- Man feeds the machine sufficiently understandable data, so that the machine can take the responsibility for the intellectual work. This is the road a number of systems, including OpenEHR, seem to take, except there is no basis for thinking the machine can take over that job just yet. In the future, yes, but today?

- Man feeds the machine the minimum of data it needs to locate and gather the resources *man* needs to take responsibility for the intellectual work. This way we would improve man's ability to work correctly, while automating away man's main weaknesses, namely memorizing massive amounts of data, and consistent attention to detail. This is the way I think we must progress for the foreseeable future.

One point I need to make clear: I have a lot of opinions about IT in healthcare, but I've also spent a lifetime almost evenly divided between the two fields that form the basis, so I don't expect it all to line up with other people's writing on the subject, in particular if they have a background in only one of the two fields. To me, the field of healthcare informatics is brimming with unwarranted conclusions and unfounded beliefs.

I will not limit myself to what I can prove, since that would make the text far too short. Far too few of the important angles have been the subject of objective research, so there isn't much to put in the bibliography, and I simply skipped creating one. It seems much of the field is built upon preconceived notions and unfounded assumptions, with mostly everyone assuming somebody else has it all figured out.

The book is organized in a couple of parts:

[1] Yes, I take that as a given.

Part I

I start out by taking a look at what the possible motivations are and could be for bothering to build EHR systems in the first place. What are they for, really? Spoiler alert: nobody seems to know.

Part II

I go into how current EHR systems work, and why they work as they do. I can't help but complain about almost everything about these systems.

Part III

Here I cover a number of problems related to building the EHR, both technical and political. Turns out that the political problems are as important, or more important, than the technical.

Part IV

I go into how doctors work clinically, which real requirements we can derive from that, and how a correctly designed EHR system built on these requirements should look. I'm making the assumption that the main goal of these systems should be to support clinical work. That is clearly not the case today.

Part V

I dedicate this part of the book to an outline of a design that comes closer to satisfying the real requirements than we have today. And with "real" requirements, I mean "our" requirements, where "our" is doctors and other medical staff.

Part VI

The last part contains a number of appendices, each of which goes into more technical detail of particular arguments or designs. Unless you are into building systems, or comparing my designs to other designs, you could probably safely skip these appendices.

Terminology

In a text like this, one has to try to be clear with terminology. It is often necessary to reduce the number of terms, even at the cost of losing some nuance, just to avoid introducing ambiguity in the text. Here follows a few selected terms I've raised to the level of "housebroken" and have used as consistently as I'm able to.

Architect, designer and so on

In this book, if not always in real life, I'll try to stick to the following roles of software builders.

Requirements Engineer

The requirements engineer takes the wishes and demands from the stakeholders (users, buyers, owners) and converts them into a list of requirements that can be used by the system architect, the designer, and the developer to create a system fulfilling the wishes and hopes of the user and buyer[2].

System Architect

I'll make no distinction between "system architects" and "software architects", that would be too much for me.

The system architect, or architect for short, is the person on the development team that takes the requirements and decides on the high level structure of the system, such as what different software applications it should consist of, which types of interconnections should be done and between what, and which platforms it will all run on.

[2] Yes, that's the idea, but in reality the buyer's wishes greatly outweigh the user's wishes. Money speaks.

Designer

The designer takes the overall structure as defined by the architect and boils it down to modules and interfaces, together with a description of what interfaces and functions should do.

Developer

The developer takes the designs from the designer and creates code. His output should be the executable deliverable.

Software engineer

This is a conflicted title, but since nobody can really agree on what it means, if it's a self assigned qualification, or an academically defined degree, which it is in some places, I use the term to describe any and all of the above roles. In other words, I include requirements engineers, architects, designers, and developers in the term "software engineer".

EHR

Electronic healthcare records (EHR) are often called "Electronic Medical Records" (EMR), but I can see no useful distinction between those two terms[3]. Also, "EMR" occasionally means "Emergency Medical Responder", which can confuse things. It can also mean "Explosive Mishap Report", confusing things even further.

"EHR", on the other hand, has a much shorter list of interpretations. It does include "Explosive Hazards Reduction", which we can only see as a positive.

In this text, I'll consistently use "EHR" to the exclusion of "EMR" for the above reasons.

Issue

We need to have a name for the reason we see patients. We can't call it a "disease", since seeing a one-year-old for a regular growth check, isn't a "disease". We can't call it a "problem", since that insults the women we see for pregnancy follow-up. I've chosen to call these things "issues", or sometimes "healthcare issues" for extra emphasis.

[3] These guys disagree, though, seeing a real difference between the terms. That won't change my mind. See: http://www.healthit.gov/buzz-blog/electronic-health-and-medical-records/emr-vs-ehr-difference/

Item

I'm using "item" in two senses. In the context of "issues", an "item" is a one-liner, a certain clinical finding with it's set of possible values. For instance, "Blood pressure" can be an item, just as "Cardiac sounds" can be one.

In many other instances I use "item" to mean what most people mean with "item". Such as "I have two items in a basket", or "are those two an item"?

His and hers

When writing a text, we always have the problem of what gender to use in third person. It's all too easy to write "him" all the time, and be revealed for the crypto chauvinists most of us older male doctors are deep down inside. But going to the other extreme, using "her" all the time, would lend a certain creepiness to the text. You can't go writing "him or her" everywhere either, since it simply looks ridiculous in the long run, and breaks the rhythm of the text, if it ever had one. Some kind of strategy is clearly called for.

My preferred strategy is to mix it up a bit. In situations where I have to choose, I'll let the doctor be female, at least in situations where the doctor comes out on top. Intellectually on top, I mean. When the doctor is described as confused, or easily distracted, or of somewhat limited ability, I'll often use "him". Patients will usually be represented as males, too.

Yes, I realize this is also a prejudiced way of thinking and writing, but I'm much less afraid of men than of women.

Acknowledgements

In no particular order, I'd like to thank the following people for constructive criticism and corrections of the text: Kim Nevelsteen, Peter Olsson, Göran Agerberg, Johan Månflod, Jack Holleran, and Ingrid Eckerman.

I'd like to thank Mary Brown of Capella University for an insightful perspective on the US medical informatics scene.

Chapter 1

About the author

There's no avoiding a section on who I am. This book is based almost entirely on my own experiences, and this both liberates me and forms a limitation on the applicability of my conclusions. There must be a large number of situations where my descriptions don't match, but I have very little other literature to base any comparison on. So I'll simply have to describe my experiences and let you draw your own conclusions as to why I'm saying what I'm saying. I'll include a lot of details, but I'll try to be brief, since this isn't supposed to be a biography, but a book about the electronic health care record.

I was born and grew up in Stockholm, Sweden, and attended Stockholm University between 1969 and 1971, where I studied mathematics, a sprinkling of programming (ALGOL, if anyone remembers that), and inorganic chemistry.

In 1971 I moved to Ghent, Belgium, where I studied medicine at the state university there between 1971 and 1978, graduated with honors. Between 1978 and 1983, I did a general surgery residency at the same university. This included 6 months of orthopedics and quite a bit of intensive care. Vascular and thoracic surgery was also part of our daily work. We weren't trained in cardiac surgery, but each of us residents assisted hundreds of coronary bypass and valve replacement procedures, and even a number of pediatric cardiac operations.

During this residency, I stumbled across an HP 2100 mini computer in a back room in the ICU. This machine had an A/D converter, a *huge* 14 inch removable disk and a fixed hard disk of the same size, each almost 2.4 megabytes in capacity. The RAM was 32k words (64k byte), and the clock frequency just a smidgen under 1 MHz. I found terminals and other peripherals in basements and other departments and hooked them up. We ended up with the teletype console which was already in place, a Tektronix

Figure 1.1: The main unit of the HP 2100A mini computer. (Photo courtesy of hpmemory.org, Marc Mislanghe.)

vector graphic terminal, and a couple of HP terminals and cheaper terminals I've forgotten the name of, and a couple of little 8" (or so) graphic displays with 256 x 256 pixels[1].

The A/D converter had 128 channels and was still hooked up to a number of bedside monitors in the ICU. It used DMA and kernel processes to write directly to disk and could read at 50 samples per second.

The entire computer was housed in two 19 inch racks bolted to the ground. When the hard disk got going real well, it would have tipped over the racks otherwise.

The way to boot this monster was interesting. Assuming it lost all track

[1] I may misremember details, so please don't kill me if that was 128 x 128 or maybe even 512 x 512, but I don't think so.

of reality, the first thing you had to do was insert a key and unlock the last 32 words in memory, so you could punch in the first bootstrap loader, the "loader loader", using front panel illuminated buttons, all in binary. After about a hundred times, I knew this boot loader binary by heart. Then remove the key, locking down the memory[2], set the instruction counter to the start of the boot loader, insert the disk operating system boot loader paper tape in the paper tape reader, then hit "run". The paper tape ran through the reader at an amazing speed, hitting the wall almost two meters to the left. Then the disk boot got going, the RTE-III "Real Time Executive" was loaded, and after a while you got the satisfying hum and clunk from the teletype.

As residents, we had to stay over in the hospital two or three nights a week, while not getting any days off, so we spent 80 or 90 hours a week at work. We slept some, of course, but never enough. Once I found this HP 2100, I hardly ever slept, instead spending my nights figuring out how to reconfigure it using "system generation", and how to program it. This is the first time I encountered the phrase "If all else fails, read the instructions", which was written on the title page of the HP 2100A manual. There's deep wisdom in that.

Using Fortran on this machine, I first wrote a system to store medical reference information. In the ICU, we had a lot of snippets of tips and tables, like how much blood a patient is allowed to lose the first hours after a coronary bypass, how long to leave a choledochus drain in place, how to calculate cardiac output using the Fick method, and so on[3]. I'd already programmed a Texas Instruments TI–59, one of the earliest programmable calculators, to calculate cardiac output, pulmonary shunts, and a few more little handy things. The TI–59 used little magnetic cards that were fed to the calculator and pulled through it by a little electric motor when you switched programs. For a while, I and the other residents in the ICU used my TI–59, and we passed it from person to person. One of the teaching staff had a PC–100A printer dock, which I used to program some statistics for a while.

With the HP 2100, I moved those calculations over to the mini computer, added a lot more of our notes of collected wisdom, and set up a terminal in the ICU nursing station. To retrieve information, I figured out an indexing system so that every page of information contained a menu at the bottom leading to other pages. I also built a subscription system so that the residents that "subscribed" to my pages could get a printout of everything that had changed since the last printout they'd gotten.

[2]Theoretically, this boot loader code should never be overwritten, since there was a hardware lock on that region, but it still happened, and I never figured out why.

[3]I'm not going to explain every medical term here, since it wouldn't add to the story. If you want to know, there's always duckduckgo.com.

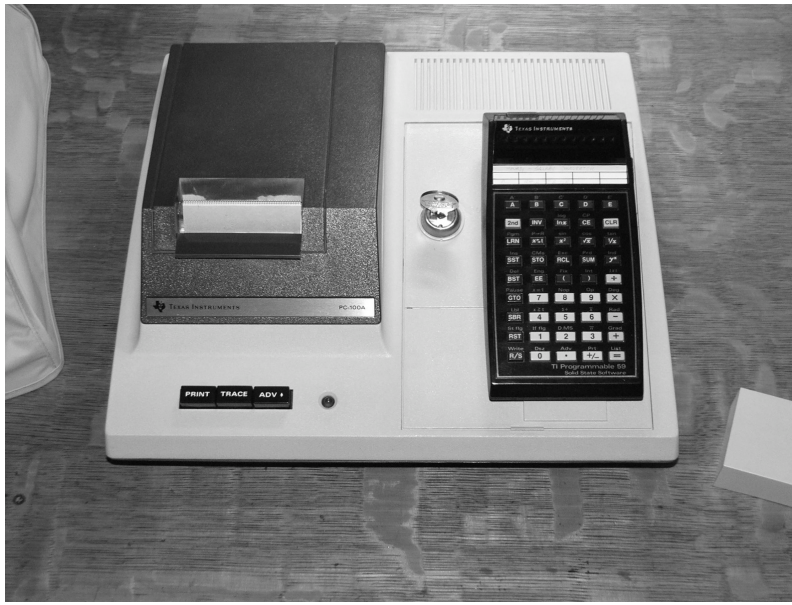


Figure 1.2: A TI–59 calculator docked to its printer. (Photo Wikimedia Commons, John Crane.)

After this, I wrote a system that used the A/D converter to connect to a bidirectional ultrasound vascular doppler machine, calculating and drawing a blood flow curve of the aorta. Together with a measurement of the diameter of the aorta, this system could calculate cardiac output without any invasive procedure. Neat and almost accurate enough for clinical use, but not quite.

I also built a tool to make researching medical records somewhat easier. This is the time before we started to disparage retrospective studies, so we did a lot of those. We researched patient histories to find relationships. Doing this was a lot of work, so I came up with a system that let you set up a list of questions to work through for every medical record studied. Then the answers were fed into the system and you could go play interactively with the data to see what correlated to what. The statistics I implemented were based on the Student T-test and the chi square test, but that was pretty much enough. The user selected two different questions from the list of questions, or data points, if you will, then the machine calculated if there was any correlation between the answers for those two questions. This took about ten seconds, and then a verdict of significance or not came up on the terminal. My dream was to let the machine run through *all* the

possible combinations on its own, but there was no way this could be done with hardware this slow, or with my less than stellar programming skills at that time.

The first study we used this for was a comparison of post surgery treatments for esophageal cancers. We thought we had quite different results from another university, and we couldn't figure out why. The very first time I demonstrated this system to another resident, Thiery Anné, the lead author on the study, I just grabbed out of thin air what I thought was a ridiculous correlation, comparing the outcome of 5-FU, a chemo therapy agent which we used a lot, depending on location of the cancer (top, middle, or lower esophagus), which I was sure had nothing to do with it. This was about the third thing I showed my friend, and we're sitting in a noisy computer room at 2 am, just beyond the last room in the ICU, the cardiology unit. The computer spit out "$p < 0.005$", and Thiery just laughed, saying "yeah, sure, as if THAT would be right... great programming, man!", and wished me luck finding the bug. I spent the next two nights trying to locate the coding error, then finally pulled out all the data, around a hundred patients, and recalculated the whole thing by hand. It all came out exactly the same as the computer had shown. That was the most important finding we had, and it was correct. It was also the last time we found anything as spectacular using that system.

During that period I also enrolled for engineering studies, and I got almost two years into it before I had to give it up. There was simply not enough hours in a day, or a night, to do all this.

We did a lot of heart surgeries, sometimes up to three or four in a day, and the main bottleneck was the ICU. Back then, a typical patient stayed on assisted ventilation up to 24 hours, then had to stay in the ICU another two days. We only had a total of six ICU beds for vascular and cardiac patients, so something needed to be done about this. Extending the ICU and hiring more nurses was out of the question. Other solutions needed to be found.

Dr. John Kirklin at the University of Alabama had succeeded in reducing the mean length of stay in his ICU down to eight hours by using computers to run the infusion pumps. It turned out that if you continually adjusted the flow of blood pressure stimulators and blood transfusions according to measured parameters, the patient achieved hemodynamic stability much sooner, and could be taken off the ventilator and moved to medium care much quicker. This is what we wanted to do as well.

I was sent to the USA with two engineers to be trained in the programming of a later operating system version, RTE-IV on the HP 1000 system, a successor to the HP 2100 system I was used to. After that two week course I went on with my wife, but without the engineers, to Birmingham, Alabama, to watch how they did cardiac surgery there.

The idea was that the University of Ghent would also invest in more recent computers, infusion pumps, monitoring, and staff, and do what they'd done in Alabama, increasing the throughput of patients through the ICU, while at the same time reducing complications[4]. But as we got back from the trip we were told that there was no money to spend on equipment. I had figured on a career as a surgeon doing some space age stuff with computers in the ICU, but it wasn't to be.

In 1983 I quit the university and took over a general practice, patients, house and all. At the same time, I started my first company which initially sold TRS–80 computer games tapes through mail order. Soon after starting the company, I wrote a very rudimentary home nursing application on a TRS–80, then got an order to write a system to estimate cooling systems for offices and computer rooms.

In the first few years of that company, I sold a number of machines, including one of the first networks (based on TeleVideo mmmOST), and also wrote an accounting program, a program to store patient record diagnostic codes, and an insurance agent management program. Mostly, writing these programs served to sell turn-key systems, which was a pretty good business back then. I also programmed and installed a few industrial control computers from Merlin Systems in the UK. Not much of peripheral electronics was available at that time, so I also designed and built some minor electronics like Whetstone bridges with differential amplifiers for Pt100 sensors, input protection circuits, and relay boards.

Around 1987 I arranged a connection to Medline[5] through some organisation I've forgotten the name of in Köln in Germany, utilizing packet switching to get there, and a search language I think was called "Daphne". When they had a particularly difficult case at the university ICU where I used to work, they called me and I did a literature search using my Medline connection, then reported back over fax with a summary of what I found. The very expensive fax machine I sold them for this purpose must have been one of the first machines used in that hospital.

I could do a search like this in about an hour's time total. I loved doing this, but it ended a year or two later when the ICU got their own set of Silverplatter's Medline CDROMs to search through. I still think my results were a lot better than what they could find on their own, though.

In 1990 I joined "Fidonet" and later ran what was then one of the larger

[4] Anything that reduces time under anesthesia, or time spent in the ICU, is known to reduce infections and other complications.

[5] Medline is the world's largest collection of medical publications, usually with an abstract, seldom with full text articles. It was the online incarnation of "Index Medicus", the paper based index of published articles. Medline is part of the US National Library of Medicine.

regional nodes[6], using three regular phone lines and an ISDN basic rate connection. Pretty bleeding edge for the time. With the computers I used for this, I also wrote a communication system to deliver lab reports and referral results to general practitioners and specialists. The system was running the "Fossil 5" communication drivers and used a system I built in the Clarion language to receive files from the labs and deliver them to users when they called in. I actually had a form of end-to-end encryption using symmetric keys that were exclusively stored in a database at each end-user's system. The client application, also written in Clarion, even contained a rudimentary medical records system with daily notes. I only charged the senders for the "stamp"; receivers got the whole thing for free.

This system was pretty darn good, but I had absolutely no time or talent for marketing, so I never got beyond about ten or 15 subscribers. In 1994 I was approached by a company called "MediBridge" who had the backing of the Belgian telecoms giant Belgacom and the University of Ghent. They had a similar product but built on Solaris systems and packet switching. The end result was way more expensive than my system with a fraction of the functionality. My existence had hindered them in several sales where there was always at least one doctor who had seen my system and didn't want theirs. So MediBridge offered me a job and some cash if I let my system die and started working for them instead. This was my way out of my general practice, so I took it with both hands.

During the next year, I was also hired by the University of Ghent to do the Belgian version of the "Episode of Care Summary" specification for electronic messaging interchange. My boss, George De Moor, was then head of the Technical Committee 251 at CEN, the European Standards institution, which led to me being an observer there for almost a year.

In the same period, I became an advisor to the ACC, an organization of around 50 private hospitals in Belgium, to help with the development of their new medical records system, the AZIS–2000. Here I learned from the inside what it is like building a medical record system in an organization that basically had no idea what they were doing, and really couldn't care less. I walked out of there alienated, angry, disappointed, and very disillusioned with vendors of EHR systems.

After a few months of writing a boring and not very good system for care facilities for the elderly, I got a job managing the IT for the department of epidemiology at the Ministry of Health in Brussels. My job there was to migrate them from Windows for Workgroups to NT, while also making them aware of a newfangled thing called "security", and things like "firewalls". A major reason for hiring me instead of any old IT manager was that the

[6] For the old-timers out there: my node address was 2:291/1906. Now, *that's* some addressing for real men, you DNS huggers!

IT people could get absolutely no respect from the doctors that made up most of the users there. The place was like a cat fight, with hissy fits all around, walls of silence where there was no screaming, and absolutely nothing being achieved when it came to computing. It was assumed that my medical background would make the users have some respect, and it worked. We did get some order in the house, better equipment, and more security. I loved that job, conflicts and all. The pay was shit, though, it being the government.

I stayed there a year, then went on to "Real Software", where I ended up in a warehouse at Zaventem airport trying to straighten out some really shitty VBA code for half a year. This is where I learned SQL Server pretty well, though. The next six months after that, I spent writing another medical communications package with some real asymmetric crypto for use in reporting in a clinical study of some new psychiatric pharmaceutical.

From Real Software, I went to C3, a small startup doing ICU software, where I wrote some of the server side code in C++ for messaging and for script execution. This lasted until the summer of 2001, when I decided to move back to Sweden.

I got a job at Profdoc AB, one of the major vendors of medical records software in Sweden. My task was to design and implement a communication system allowing transfer of referrals, reports, lab results, and electronic prescriptions between any number of large and small client systems. If you're keeping count, this was the third such system I developed. This one was also based on asymmetric crypto. While my first system was based on Clarion and assembler, and my second system on C++, this system was entirely written in Borland's Delphi, since that was the house language. It took me a little more than three years to build this system, and, as far as I know, it's working just fine still.

In the years that followed, I went back to work as a GP part time in Sweden, while also developing a few minor applications. In 2008 I got hired as a contractor to develop a model application in C# for another division of Profdoc. The idea was to set up an architecture that they could follow when rebuilding some of the legacy applications they had in house. I did this for about one and a half years.

In april 2010, I suddenly got the idea of how a medical record should really work, an idea so different from how the EHR systems of today are designed that I simply had to develop a prototype just to see if it could be real. I made the first version of this system which I call "iotaMed" ("Issue Oriented Tiered Architecture for Medicine"), for the iPad in Objective-C. Ever since then I've been refining the idea and writing about it in different media.

I thought the advantages of my iotaMed would be obvious to everyone, but I was wrong. Doctors, in general, understand the idea and find it

obviously better and entirely a new thing very different from what we have with current EHR systems, while everyone else, that is administrators and developers, can't see neither the difference, nor the point. I keep getting the remark that "we already have that", or "nobody ever asked for that". This attitude is both infuriating and enlightening. It clearly explains why the systems we keep getting from these developers are so useless to us. It's as if our (the doctors') way of thinking and working is so alien as to be invisible to them.

I then did what developers always do when met with a seemingly unsolvable problem; I introduced another level of indirection, or as other people say, took a step back. I decided I first need to explain what the problems are before I can offer the solution. You can't answer a question that hasn't been put yet.

So this brings us to today and this book. It is intended to provide the right questions for a change, and maybe a few of the possible answers. I will only touch on the iotaMed design occasionally, since that isn't what this book is about.

Part I

The basics

Summary

Before going into what's wrong with current systems, or how to build the next generation, we have to look over the basics of medical record keeping. These basics include identifying the driving business models, the stakeholders, the motivations of the stakeholders, and how doctors are trained and keep up to date. It also must include the history of the medical records, and an outline of the relationship between the medical record and the professional, in other words, how does the medical record actually help the medical professional do a better job?

Chapter 2

Why are we doing this?

Before we embark on the road to revolutionize healthcare with information technology, we have to know what we mean by "revolutionize". Yes, we want to make healthcare *better*, but what do we mean by "better"? I'll present a few popular, and sometimes misguided, reasons for doing this.

You can't have it all, at least not immediately or at the same time, so you'd better choose which of the following goals you prioritize. That's very important, but it is even more important that you clearly express what you just decided. There's nothing worse than having a bunch of otherwise well-meaning people trying to achieve contradictory goals together, while none of them is aware that they are working at cross purposes with each other.

2.1 Making people healthier

One goal could be to make people healthier. It sure sounds worth-while, but stop for a moment to think what it means. For instance:

- Avoiding disease.
- Improve coping with disease, and curing it.
- Improve performance of otherwise well people.
- Improving the basic health of the next generation.
- Avoid making mistakes in healthcare provision.

And so on for a very long list.

2.2 Quicker diagnosis and treatment

Computing could improve time to diagnosis, and time to treatment by better scheduling, automated procedures, and so much more.

It could also mean implementing robotics, so that patients don't have to wait for a real doctor to arrive. Or it could mean implementing telemedicine so that the doctor won't have to travel to the point where his expertise is needed.

But mainly, "quicker diagnosis and treatment" builds upon the other elements I discuss in the following.

2.3 Simpler

Properly designed software could eliminate a lot of complications in the provision of care, making more time and effort available to doctors and nurses to spend caring for the patient. If done right, it would take care of tedious and error-prone details in the healthcare process, but if done wrong (as it mostly is), it will instead add tedious and unnecessary steps.

> *If your automation adds steps to the healthcare process without significantly improving on its results, we're better off without that automation. Improving on management information without provable effect on the direct care of the patient is a misuse of resources and should be avoided. The provision of management information cannot usefully be a primary goal of healthcare.*

Interestingly, the current generation of EHR systems are so bad that the most common user requirement is "make it simpler", which practically always means "make software I don't have to interact so much with", or in short, "leave me alone!". That is a low sad level of ambition for system design. For many, or most, doctors and nurses, EHR systems are seen as a chore, taking time from "real" healthcare, instead of being of assistance in their work.

2.4 Safer

Using computerized processes could ultimately mean less human mistakes, thus making healthcare safer. Since computers are programmed by humans, we can't really eliminate mistakes altogether, but the potential is there to reduce them, at least.

There is also an immense potential to screw things up royally. Paper records are rarely entirely lost during a care episode, i.e. while the patient is under active care, and when it is, it's troublesome for the care of that

one patient. But if a central EHR system goes on the blink, *everyone's* record is lost in its entirety. So even though EHR systems can reduce the frequency of losses, each single loss can completely outweigh any of that, nullifying (and then some) the gain in "safety". These major events may be unpredictable, but they aren't any less certain to occur because of that unpredictability.[1]

In short, the requirement for "safer" healthcare, could be truthfully be amended to "not making it too much unsafer".

2.5 Analysis of what we're doing

Theoretically, if everything we do to a patient is recorded into computers, it would be easier to analyze it after the fact, enabling new discoveries and finding process problems. Spoiler alert: automating science this way doesn't work. I'll discuss this in the chapter on researching the EHR (chapter 13).

This type of analysis of the EHR could also be used to determine the efficiency of the healthcare delivery process, akin to factory productivity. This is something managers love and healthcare staff usually don't, but it fits in with a certain class of management theories[2], so it's probably here to stay.

2.6 Economic tracking

Having all medical information in a computer system ought to enable more exhaustive and more accurate tracking of costs, leading to better allocation of budgets. But economic tracking should never be the primary, or even most important, reason to introduce information technology. As a corollary: never let non-medical management specify or purchase automation technology for healthcare purposes, since the technology will be designed to work for them, not for healthcare staff or patients. A second corollary says that this explains why our current systems suck. They're not made for healthcare, they're made for healthcare management.

2.7 More transparent to the patient

Having all this information in databases ought to make it easier to allow the patient to see exactly what has been done and what has been found.

If this is your motive, you also have to be clear what you expect to change by empowering the patient. Do you expect them to make better choices in

[1] I strongly encourage reading Taleb, The Black Swan, to fully appreciate this point: the exception *is* the rule.

[2] "New Public Management" is a major threat, but there are doubtlessly others.

what healthcare they need, which healthcare providers they should consult, or do you expect them to directly or indirectly perform a quality control function?[3]

If your patients have none of those possibilities due to political limitations in your healthcare system, then giving them access will not result in improvement of the process, only in frustration. On the other hand, giving them access may cause them to hate the systems enough to become more politically active, a good thing in itself.

Also see my section on encapsulation (chapter 24).

2.8 Reduce accidents

Using a computer to order diagnostic tests and start treatments could enable us to automatically check for and prevent common errors. For instance, detecting contra-indications while prescribing could reduce the number of deaths due to medication errors. The system could also warn for missing steps in a diagnostic or therapeutic routine. It could also draw our attention to dangers specific to a particular patient, so we don't order an MRT on a patient with a pacemaker, for instance[4].

2.9 Increase consistency

If the computer is used to propose or track the progress of diagnostic or therapeutic procedures, it could also remind the users of all the generally agreed upon steps in these procedures, ensuring a more consistent quality of care.

IT systems can also be employed to increase the consistency of the notes taken and data collected. This makes the notes look good on paper and screen, but it's doubtful if it has a value in and of itself.

Increasing consistency in delivered care, however, almost certainly has great value. If we decide on a particular optimal line of diagnosing and treating a certain well-defined group of patients with a particular well-defined problem, then we should ensure that all these patients receive the best care we can give, by ensuring they are all consistently treated according to the relevant guideline. Also, if problems occur, then they are more easily observed, and more easily corrected, if we already have a consistent application of our best knowledge. One should take care, however, not to apply particular guidelines to the wrong patients, and this is harder than it seems. This is where the "clinical experience" of doctors is really needed.

[3] What would that be? Rating the performance of the doctor? Like Facebook likes? Yikes.

[4] Unless you think peeling the patient off the insides of the MRT magnet is a hoot.

One shouldn't let guidelines turn into cookbook recipes to be mindlessly applied regardless.

In other words, consistent documentation may or may not mean much in the delivery of care, even though it may look good to administrators. Consistency in care, however, is a worthy goal with many positive outcomes.

2.10 Reducing need for doctors

A well-designed computerized medical record could replace doctors in some situations, if the system was provided with guidelines and handled by a competent paramedic. The question is how far this automation could safely be exploited.

Many civil servants dream of the day they'll be able to automate away these difficult and expensive doctors. Once they do, they'll probably have demanding and difficult IT systems in their place, since much (albeit not all) of the difficulty stems from the problem domain itself, not from the healthcare workers in that domain. For a layperson, that can be a very difficult thing to distinguish, however.

2.11 Playing with computers

Introducing computers could be a goal in itself. There's always some love of technology for its own sake. Most systems designed and developed by doctors for their own use, falls under this heading[5]. These systems often have very clear advantages in workflow, but often fall down on the technical design and implementation[6].

If the presence of computers increase the attractiveness of the institution to potential candidates, the introduction of these systems would indeed have a value in and of itself, but as computers are increasingly common, this reason for automation is rapidly becoming irrelevant[7].

2.12 Agree on the purpose

It doesn't really matter, in the grand scheme of things, what the exact reason is why we do a healthcare related IT project, as long as two conditions are fulfilled:

1. *We* know the reason, and we've articulated it clearly.

[5] Yes, that includes me.

[6] This clearly doesn't include my systems.

[7] Particularly since most people nowadays have much better systems at home than those at work.

2. *Everyone else* involved in the same project is on the same page, and understand the primary motivation for the project. It simply *must* be clearly communicated and agreed upon. We need to regularly reconfirm that this is still the case, as the project advances. If there's "mission creep", that is if the goal is slowly changing, everyone on the team must be made aware of that and adjust accordingly.

There's nothing worse that having a group of talented people working at cross-purposes, each with another idea of what the goal is, and consequently getting nowhere. Everything we do in such a group will by definition be a failure to at least some, often even everybody, in that group and outside it.

Chapter 3

The business model

There's a business model behind every decision on how to structure healthcare and its support services. That business model could be based on the overall cost to society of healthcare issues, versus the cost of providing relief and prevention. These are the concerns at a large scale, by region or nation. On a smaller scale, the business model could include measures intended to displace costs, such as moving them to other actors, and measures to optimize handling of cases.

Let's take a few examples to illustrate the difference between the large-scale business model and the small scale business model.

3.1 Large scale business model

If we calculate the cost to society of a case of polio, including treatment costs, assistance cost, and loss of productivity, it's much more expensive than the alternate cost of prevention[1]. It's obvious to anyone, except maybe to the anti vaccination nut cases, that global vaccination and ultimate eradication of the virus, is the way to go.

The same calculation has also been made for certain types of cancer where we have useful screening methods, such as for breast cancer, and colon cancer, and early detection and treatment is the winning proposition, even without considering the human cost of contracting cancer and having it detected too late for curative treatment, in the first place.

We can also show that the correct and effective treatment of joint diseases, diabetes, hypertension, vascular disease, and a host of other problems, is a generally good idea from a national economics standpoint. All these things cost less if treated according to the state of the art. Even

[1] http://www.polioeradication.org/Portals/0/Document/Resources/StrategyWork/EconomicCase.pdf

bleeding edge, and often extremely expensive, treatments are economically defensible once we take the evolution of the treatment into account over a period of time. Many of these initially expensive treatments lead to much more affordable and much more applicable treatments in the future, treatments that would never have been developed if the initial expense wasn't made.

In short, paying for the best healthcare you can provide to your population is one hell of a good investment for any nation, even without considering the human cost of disease. No civilized nation would argue otherwise[2].

Clearly, if healthcare is managed primarily according the the benefits to the nation as a whole, the systems developed for healthcare will focus on providing prevention and treatment from a medical perspective. The management of local expenditures will still play a background role, while the calculation of "profit" makes no sense. There's no immediate and local payback for each treated patient; the payback is on a national scale and in a longer perspective.

3.2 Small scale business model

The small scale business model comes into play for a hospital, a department, or a practice. It's all the rage in models like New Public Management (NPM), where departments are made into cost centers, and each is incentivized to increase profits and reduce costs, on the assumption that if all departments become more cost conscious and more profitable, the organization, even the nation, as a whole benefits.

This model has been used for a long time in industry, and is becoming increasingly discredited, especially since the most profitable information technology companies are leaving this model. Companies are increasingly seeing the benefit of having all departments work for the common good of the company as a whole, instead of artificially competing with one another. Healthcare seems to be a few decennia behind on this learning curve.

When applying this cost center model to healthcare, at least in Sweden, a fictional cost is assigned to diagnostic and therapeutic actions. Every medication prescribed is assigned the retail cost and "charged" to a fictional budget the prescribing organization has. Every radiology exam, or lab order, is similarly "charged" to the provider that ordered it. Every visit from a patient ends up on the "profit" side. The idea is that providers should be incentivized to save on costly examinations and treatments, while at the same time seeing as many patients as possible.

[2] Except the USA, but you guys are slowly getting the message, too.

In this whole arrangements, two things are missing. First, referrals are "free". Secondly, there's no "reward" for actually making patients better. See where this is going?

Yes, indeed, in order to keep organizations in pretend money[3], the trick is to refer patients to someone else if it looks like their treatment is going to "cost" more than their visits bring in. On the other side of the referral, the referee is incentivized to refuse as many referrals as possible, especially if they look like they'll need a lot of expensive care. Or, alternatively, they're incentivized to only accept referrals where the originator of the referral has done as many expensive diagnostics as possible *before* referring the patient, on her own "dime". This turns the referring procedure into a confrontational game, instead of the cooperative effort to help and serve patients it should be.

Just to take an example: orthopedic surgeons take a lot of x-rays, naturally. They need them in most, but not all, patients. X-rays are subsequently a large "cost" item in their budget, and they'd love to get patients referred who already have all the needed x-rays taken beforehand, because of this. In the province I work, they've taken this to the logical extreme. The department of orthopedics at the university hospital doesn't even accept a referral without accompanying x-rays. This causes a lot more x-rays than necessary to be made, but not on their "dime". The result is a hugely increased total cost of healthcare, longer waiting times for other types of x-rays that are really necessary, and a loss for everybody's budget, *except* the department of orthopedics, which looks on paper as if it is efficient.

Other examples I've personally experienced are the emergency department sending patients to primary care for writing prescriptions, cardiologists referring back patients with a recommendation for the referrer to prescribe expensive medication the cardiologist prefers but can't "afford", and more of that kind. In each instance, the patient is given a run-around, while the *total* cost of care goes up, and the originator of the problem is praised for being budget conscious. It's important to remember that the Swedish healthcare system is firmly "single payer", so all these costs are ultimately covered by the same agency, the state.

If healthcare is provided under a small-scale business model such as NPM, the supporting systems become very important to calculate and control costs, while also keeping tabs on profitability. Any benefits on a national scale or longer term will become invisible and ignored. This explains why there is *no business case* for better healthcare under these ideologies, only in local sub-optimizations. Small business scale practices make sense in small

[3]The departments with most pretend money left and the end of the year, get preferential treatment when it comes to staffing and costs the year after.

businesses, make very little sense in major corporations, and are absolutely toxic to large scale population concerns such as healthcare.

> *Finally, we have to note that regardless of budgets, the tax paying citizens may want to pay for healthcare, even if it wasn't economically efficient. Human wellbeing consists of more than economically measurable aspects of life. It's absurd to only invest in measures that result in a financial return on investment. If that was all we were concerned with as humans, we wouldn't procreate.*

So, how come patients aren't worse off than they are in Sweden? With a model like the one I just described, one would expect the healthcare to be really crappy, but it isn't. The explanation for that lies in a little bit of compensatory socialism; doctors aren't paid according to the outcome of the budget numbers, or the number of patients. Doctors, in general, receive a monthly fixed salary, making them independent of the NPM games. So even though it's a real hassle having to argue with the orthopedics department, and others, over referrals, doctors have no incentive to save on these fictional charges, usually preferring to get the patient taken care of, regardless.

In short, it seems we're lucky that doctors don't need to care about the NPM numbers and budgets, partly neutralizing the best efforts of public management to corrupt and destroy the healthcare delivery process. One of two things can happen in the future: either the whole NPM idea is scrapped and replaced with a grander scale motivational system that measures and rewards healthcare outcomes, as it should do, or doctors move off the fixed salary regiment and become rewarded according to the NPM based measurements, including small scale monetary rewards, moving Swedish healthcare to a similar system as the USA and almost certainly making healthcare as inefficient, but also highly unequally distributed as it is there[4].

[4] http://www.nejm.org/doi/full/10.1056/NEJMp0910064

Chapter 4

What are doctors made of?

Naturally, we're all made of some calcium, water, neural cells, and not an inconsiderable amount of intestinal content, but what I really want to talk about is what kind of knowledge and training is necessary to "build" a functioning physician.

The knowledge we need can be divided into theoretical knowledge of the healthy human, knowledge about the mechanisms of disease, dexterity in clinical examinations, craftsmanship in diagnostic and therapeutic procedures, and current knowledge about diagnoses and therapies for a range of problems.

4.1 Theory of the healthy human

The first courses at medical school are all focused on teaching the basic normal functioning of the human body. This includes stuff like biochemistry, anatomy, physiology, and more. As years go by and science advances, this body of knowledge tends to increase rather rapidly. We have long left behind us the time when a doctor could more or less have a grasp on what we know of the normal human body, so we have to assume that the practicing physician will only have at his fingertips the most rudimentary facts about the body. Any clinical work that requires a more detailed knowledge must also be supported by tools that provide the clinician with the knowledge needed.

Except for the occasional old anatomical atlas and well thumbed biochemistry book, you won't find many such tools in most clinician's offices, simply because very few of these tools exist. Even if they did, the clinician usually has just the one computer for the medical record, and those tools either don't work on that system, aren't allowed to be installed by misdirected IT support staff, or don't work together with the existing medical

record software in any case.

4.2 Mechanisms of disease

Knowledge about mechanisms of disease is rapidly evolving. It's evolving so fast that any course knowledge the doctor may have is quickly outdated. There is a need for constant education about these developments, not on a yearly basis, but on a monthly or even weekly basis.

4.3 Clinical examinations

Clinical examinations encompasses routines like taking a blood pressure, listening to the chest sounds (and recognizing what you're hearing!), palpating the abdomen, testing reflexes, evaluating joints and tendons, and so on. *How* to do most of these clinical examinations must basically be taught in person, during classes or during work in the clinic with a real patient and a tutor. Some of these examinations can also usefully be taught to an already experienced physician with the aid of diagrams and explanations. An example of that can be seen in the excellent webpages from the American Association of Family Physicians (AAFP[1]), about the examinations of the shoulder joint, a pretty complex subject (see figure 4.1).

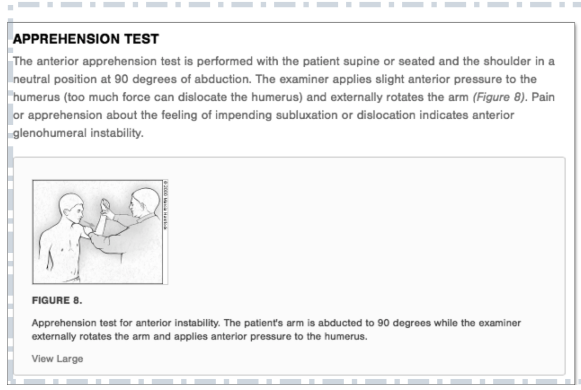


Figure 4.1: A clinical examination, as shown on the AAFP.org site

The illustration looks as if it came out of a jujitsu manual, but actually demonstrates one of the important clinical tests for instability of the shoulder joint. It is described in text and with an image how to perform the

[1] http://aafp.org

test, and the meaning of a positive test (anterior glenohumeral instability) is also mentioned in the brief text.

Since I'm not a shoulder specialist, I'd be hard pressed to remember this test, how it's done, and what it means. I've got hundreds of other clinical tests to remember, and this is one of those I keep forgetting, and I'm sure I'm not alone in this.

What I'd need in practice is a quick reference that pops up when I have a patient with shoulder problems. That reference should include this and other relevant tests, so I know what to do. As it is today, I have to know about the AAFP website, know where to look and go find the test. Not only that, I'll also have to describe the test, at least briefly, in the medical record together with the result. If I don't, a future reader of the patient record will either not know what I'm talking about, or will have to look up the test and the meaning anew, and this process will have to be repeated each time the record is read.

Clearly, what we need is a direct link between the documentation for how the test is done, with the result of performing the test on the patient, and have all that become a part of the patient's medical record.

4.4 Craftsmanship

Craftsmanship comes into play when it's time to draw blood, open the abdomen, hack off limbs, check the eyesight or retina, or aspirate earwax. Depending on speciality, this craftsmanship can go from the very limited, as for psychiatrists[2], to the very extensive, as for surgeons.

Most, if not all, of this craftsmanship must be learned the same way ironsmiths learn their skill, in a master-apprentice setup. When these skills need updating, doctors again go watch each other do whatever it is that is being done, then try it themselves under initial supervision. There is some help that IT systems could provide for this process, but nothing earth-shaking[3]. This process probably needs to stay the way it is for the foreseeable future.

[2] Yes, I realize that making people talk and then listening to them all day is a skill too, but for some reason, I'm not counting that. Practically all of a psychiatrist's skill lies in other areas than the purely manual.

[3] There's a lot of hay being made of computer simulations for training surgeons, but it's out of proportion to its relative importance, at least compared to other urgent problems in medicine that need computerization. But there's more money to be made more quickly selling simulators, I guess. Or, just as likely, it's easier for journalists to understand.

4.5 Diagnostic and therapeutic knowledge

Knowledge about diagnostic tools falls partly into the same bucket as "clinical examinations", but it encompasses much more. Knowing which radiology studies to order for confirmation of exactly which diseases, is a skill that needs constant updating. Knowing which lab tests are available and what they mean, also includes some considerable work to keep updated. Even knowing which diseases *can* be diagnosed using available tools is far from simple. Worse, even knowing which diseases *exist* is a challenge[4].

The therapeutic arsenal, i.e. the number and kind of methods we have at our disposal to alleviate diseases, is changing at a fantastic pace. There is no way a practicing physician can keep track of even a fraction of what is going on in this area, except for a tiny part of a single speciality. Even knowing which other speciality to refer a patient to, or even that the patient should be referred at all, is complicated enough to be regarded as a knowledge area in itself.

The sum total of all diagnostic and therapeutic knowledge is so vast, and so quickly changing and expanding, that it's impossible to rely on human memory alone. There's not enough hours in a day for any doctor to even barely keep up. This is where we need knowledge based tools the most. But to make them even remotely useful, they have to be linked to the medical record in such a way that searches are triggered and enhanced based on the information about the patient in the records, *and* that the results from such searches also becomes part of the patient's information, including the conclusions the doctor drew from the searches in view of the patient's particulars.

Having the searches of the universe of knowledge unconnected to the patient record not only increases the cognitive load on the doctor beyond the bearable, but also loses a large part of the advantage, since the relationships found, and decisions made by the doctor cannot be persisted into the record, so they can't be used as a base for future analysis and decisions. The waste of human intellectual effort is simply epic.

[4] Which, by the way, is far fewer than the internet would have you believe.

Chapter 5

The History of medical records

As in all books, there is this "history" thing. But in this case, the history is essential to understand why things are as bad as they are.

In what follows, remember that I'm old enough to have actually lived through the described evolutionary stages myself.

5.1 The absence of records

Not long ago, some general practitioners had no medical records at all. When I first took over a practice in Belgium in the 80's, the "records" I got consisted of two stacks[1] of documents:

1. Letters, lab reports, and other documents that my predecessor had not yet seen and discussed with the patient, in reverse chronological order. In other words, they were dumped on top of each other as they came in.

2. The same kind of documents, after they'd been seen, in order of their processing. In other words, they were dumped on top of each other in the second heap once seen and discussed with the patient.

The system worked as follows: the patient comes in and asks what the specialist said or what his blood tests showed. The doctor then asked around what time the visit to the specialist occurred or the drawing of blood, then proceeded to locate the document in stack number one. After reading it and discussing it, he prescribed something or other and off the patient went. The document ended up on stack number two and was never

[1] Literally stacks: one of them was a meter high, the other a third of that.

seen again. The whole incident then lodged somewhat loosely in the memory of the doctor and hopefully more permanently in the memory of the patient.

Obviously, this system can only work for a limited number of patients, and only if they stay with the same doctor for a long time. This kind of patient-doctor relationship was common back then, but is rapidly becoming extinct as both doctors and patients become much more mobile. Also, the number of doctors, both general practitioners and specialists, involved in the care of a patient is increasing, resulting in the number of contacts between each doctor and any particular patient is going down.

But the worst aspect of this old type of doctor-patient relationship is that it is dangerous. Important details are forgotten, and the patient's history can't effectively be transferred from one doctor to his successor.

5.2 Paperbased mementos

Obviously, this was a terrible state of affairs. Around this time, most GPs in Belgium[2] started keeping a real medical record for two reasons:

1. Fear of lawsuits. If you're sued for malpractice and you have no records at all, you're doomed.

2. To memorize details, such as exactly which medicine was prescribed for exactly what period of time, exactly when. And blood pressure measurements, and such.

In other words, the paper record evolved to record hard to remember details in diagnosis and treatment on the one hand, and simultaneously to a log of all interactions with the patient for legal reasons. The knowledge about the patient as such, his diseases, preferences, and most of all the overall plan in the diagnosis and treatment was not so much written down as memorized by the doctor. After all, the patient always went to the same doctor anyway, so why write it down?[3]

The key thing to remember is this:

> *The classic paper based medical record was only intended to support the family doctor in the maintenance of hard to memorize*

[2] I'm talking about Belgium for two reasons: firstly, that's where I was. Secondly, Sweden had proper records for patients much earlier at least in hospital care and care centers ("Vårdcentraler"), while genuinely independent GPs are a rarity here in Sweden and I don't know how they handled records back then. The first general practice records were created in Sweden in the 18th century, and became commonplace in the 1960's or 1970's.

[3] It was also a great incentive for the patient to stick to the same doctor.

details and was never designed to contain the overall picture of the patient or any diagnostic or therapeutic plan. Since there is no assigned and permanent doctor in most practices anymore, it makes no sense to automate the paper based records without considerable adaptation to the new medium. But that is exactly what has happened. The electronic record is designed as if it was a paper record, but in digital form. All the opportunities for improvements that the digital form brings have been missed. Current medical records are characterized by a torrent of useless details without a unifying context.

Chapter 6

The stakeholders

A number of different professional groups have a stake in how the electronic healthcare record system works. Their primary use of the system differs, and their demands on functionality are often in conflict.

Physicians mainly use current systems to record and retrieve patient histories, and to send and receive referrals. Other important uses are the creation of prescriptions for medication, orders for radiology and other technical diagnostic procedures, and retrieval or radiology reports and lab reports. The system is also used to create correspondence, such as letters to patients.

If the physician works alone or in her own practice, she'll also handle billing and payments through the system.

That is what the physician is doing with current systems, but a lot more should be done, if it was only supported by the system, not least support in the management of diseases and other issues.

Nurses use the system to find out which diagnostics and treatments the physician has ordered, and to take notes on the progression and results.

Administrators mainly use the electronic healthcare record to measure the production as number of patients, severity of caseloads, how many beds are free, and more such. It is in administrators interest that as much data as possible is coded in a way that allows statistical analysis of the organization and its throughput.

Since there are several groups of stakeholders, all the groups should have their interests represented when designing the systems, but in reality the interests of the management group not only overshadows the interests of the clinical groups, but is also growing. As a result, these systems increasingly turn into management systems for healthcare where the role of doctors and nurses, as far as the system is concerned, is increasingly reduced to being data input clerks. There's very little, if any, effort to increase the utility of

the systems for better diagnostics and treatments.

Chapter 7

How should the EHR assist us?

It's useful to compare the tools and methods we use in medicine with how other professions evolved. All professions have in common that they use knowledge, methods, and tools to achieve a goal. Medicine is more lopsided than most professions, lacking many of the tools we need for optimal performance. Not so coincidentally, it is unique in the sense that most tools for medical professionals are specified, designed and developed by lay people, while most other professions aren't that unfortunate. Pilots have a say in how cockpits should look, architects have influence on the software they use for design, but doctors often get medical records software that no doctor would have specified. The result is clear defects, entire missing areas of coverage, and we need to look into what those missing areas are.

7.1 Compare to other knowledge areas

We don't have to make a choice between a system that assists us in decision making and a system that documents our actions, since the very process of working through a problem using knowledge support can be automatically documented, and in itself covers a major part of that documentation. One could argue that any documentation that covers a part of the medical events that are *not* part of a process, that is of a decision tree, except the patient's subjective history, by definition has no consequence and is thus of less value. The exception being, of course, parts of the medical process that are not covered by the tool currently used for knowledge support, but which should have been.

With this reasoning, we arrive at the conclusion that any free-form medical record notes are a symptom of defective knowledge support functionality, which is a good description of *all* of our notes as done currently, since there is no knowledge support functionality at all worth the name in any

of our systems.

To see how the EHR fails in assisting doctors in their work, we need to compare it to other known processes that do work much better, and the example I'm choosing is "fixing a computer", since that is what most of us do far too often.

If we put the process of "fixing a computer" right next to the process of "fixing a human", we can find the same five stages or phases in both processes:

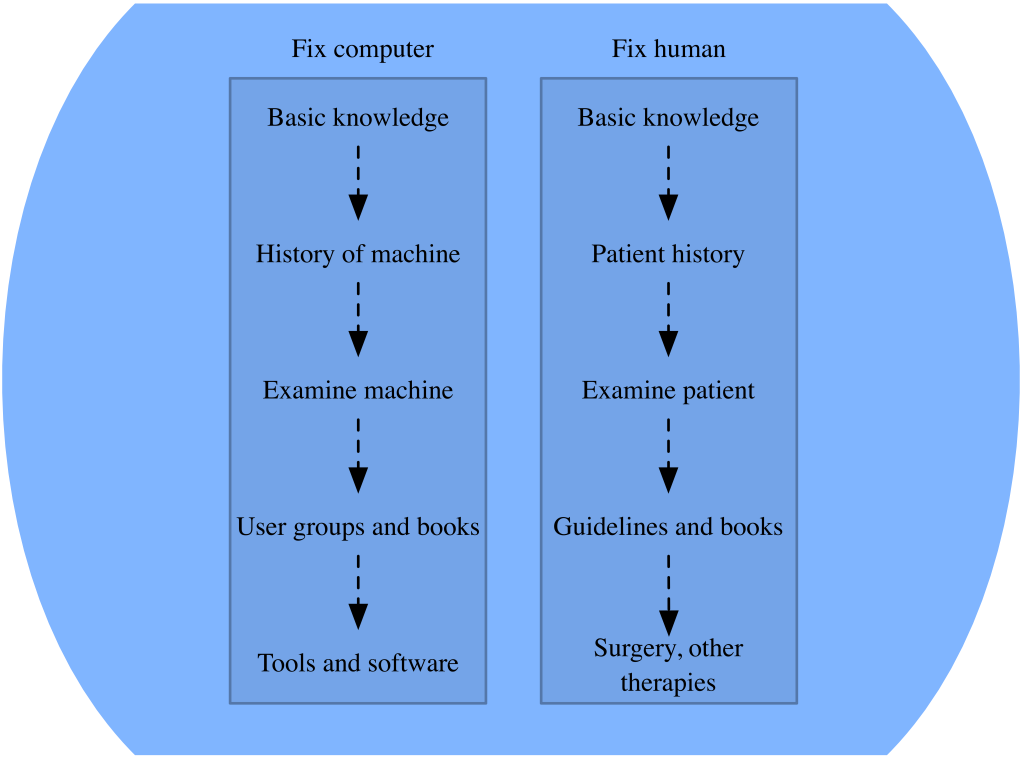


Figure 7.1: Comparing how to fix two similar things.

The first part, "basic knowledge" (of computers), has the corresponding "basic knowledge" (of humans) in the other column. These requirements are somewhat analogous for the two knowledge areas.

The second step, the "history" part is the only thing we have developed to a significant extent in EHR systems. Similar to the "computer fixing" process, this is not the most important step. It's nice to have, sure, but surprisingly easy to live without.

When we get to step three, "examine machine" vs "examine human", we find support for some, such as radiology, lab, and most therapies.

The real "meat" of the process to fix a computer is in the fourth step: "user groups and books". Not many computers would be fixed today without the ability to reference what other people have seen and how to fix it (or not, as may be). Trying to fix a computer without any reference to these resources is doomed to failure except in a few trivial cases.

Actually, the same is true for medicine. Since we don't have the same kind of easily available resources, medical practice is still in the 1980's if we compare it to fixing computers. We're still offline, so to speak. We can fix humans, but the only knowledge we can use is what was hammered into us at medical school, or that we can fortuitously remember from a more recent CPE class.

That fourth step, the step that really makes a difference, is all about gathering recent knowledge about a problem and its solution. That part is missing entirely from the EHR systems we have today. Yes, those resources are available elsewhere, but they are both hard to locate and hard to use *while examining and treating a patient*, the very time when these tools could make the most difference.

> *Let's make another comparison: the architect's productivity has certainly increased with the introduction of email and software that allows him to both write and maintain textual documentation, but the real advance of architecture is enabled by CAD software. Where is the CAD software equivalent for medicine?*

Since our EHR systems are specified by civil servants, it's only natural that any provision for providing better healthcare, quicker diagnostics and more consistent therapies, have been left out entirely. None of these things matter to the administrators, so they simply don't care to require any such functionality.

> *You get what you pay for, or rather, you get what he who pays for it asks for. And healthcare functionality is not part of that.*

7.2 What should I do?

Building on the example in the previous section, we can translate that into the steps that follow.

The first thing the software should help me with is a guideline on basic stuff, such as:

- What history elements must I query the patient about?

- Which clinical signs and examinations are basic to most problems and should be considered[1]?
- If I have a working hypothesis, the software should present me with signs and techniques that can be helpful to confirm or exclude that hypothesis. It should also present the major criteria for a diagnosis, without drawing any conclusions automatically (see my discussion about active software (chapter 25)).

7.3 How should I do it?

When I do a clinical examination or determine clinical signs, the software should:

- List the signs and examinations related to this, their names and a list of possible outcomes.
- Give me an informational page showing me how to perform the clinical examination.
- Give me information about what the clinical examination results mean and imply. Included here should be pointers to other examinations that could be valuable to complement the result.

If I decide to order a diagnostic test in the form of lab tests or technical diagnostics such as x-ray or ultrasound, the software should assist me as follows.

- Show which diagnostics are available.
- Show what prerequisites apply to the diagnostics, and when they shouldn't be performed (contra-indications).
- Show the relative cost of the diagnostic, both in resource use and in costs to the patient (radiation load, pain, risk for complications, time, etc).
- Show *where* the test can be done, and what provider to send the order to.
- Help me fill in a form for ordering the test, including all the elements the intended recipient of the form has determined is needed for the order.

[1] "Considered" means just that; you should think about it, and do it if you think you must. The point here is that if you don't do something in a guideline, it should be because you had a reason not to do it, not that you simply forgot.

- Helpfully propose the right documents from the records to include with the referral.

7.4 What did I forget?

The application should point out to me what I forgot, such as:

- Which other diagnoses should I consider and exclude for this patient? That is, present me with a list of "differential diagnoses".
- Which lab tests or diagnostic tests have I forgotten to perform to confirm this tentative diagnosis?
- Which therapy have I forgotten to start, or stop, for this diagnosis?
- Which reporting have I forgotten to perform?

As before, the system shouldn't tell me what to do, only what to *consider*. As a user, I should feel comfortable that I've considered all the angles and that my choices are made not from ignorance, but from weighted judgement of all the elements.

7.5 History in context

When retrieving the history of the patient, the EHR system should present a list of issues for the patient, and all notes, conclusions, examinations, and other documents *in the context of their respective issue.*

Part II

Current systems

Summary

In these chapters, I'll describe examples of current electronic healthcare record systems, and how we work with them. I'll also describe which knowledge support systems are available and make a few points about their usefulness, or lack thereof, and try to identify what is missing to make them more useful.

I'll also describe how a doctor works, and how she interacts with the electronic healthcare record, including the limitations due to time and place of use.

Finally, I'll go into the information model that is used in current systems, and what's wrong with it.

This part is mainly catering to the non-physician reader, providing the background necessary to understand why our current systems don't satisfy the real needs of doctors, and by implication, what needs to change. Doctors are also encouraged to read these chapters to see the assumptions and the background I have when writing this text. Contexts vary, and yours may be different.

Chapter 8

The goal of the system

The goal of the system should be to help the healthcare professional do a better job. Some functions of EHR systems support data entry and communication; functions which are generally fairly well developed in current systems. What is almost entirely lacking, however, is knowledge and process support, as I described on page 47. Some simpler processes for nursing can be found here and there, but nothing really significant is going on.

Note well: I'm not saying there are no knowledge or process support initiatives out there, but what I *am* saying is that there are no significant such initiatives that are fully a part of the EHR. There's a great number of such stand-alone initiatives, fragmented and each covering just a part of the problem area. But even if there were wall-to-wall coverage in such a tool, it's still not a part of the healthcare documentation process. If you have to interrupt the regular documentation process to go look up something, and what you find does not automatically become documented, it's not only harder to do, it also does not enrich the documentation with any reasoning resulting from the knowledge you looked up. It's not enough to define in the documentation what you do to the patient, it's essential that you document *why*, or *why not*, you are doing it, and that motivation is lacking if the knowledge support is not integrated into the same tool.

One could argue that the EHR has only a documenting function, and that the knowledge support function should be separately implemented and provided, but that would by necessity imply duplication of effort and redundant information. One could also argue, as I do, that the EHR has erroneously been viewed and developed as a documenting tool instead of a supporting tool. Current systems are also increasingly subverted to become data gathering tools for management purposes *instead* of a tool for healthcare provision, due largely to a power grab of administrators in healthcare.

A nice illustration of how badly current systems are conceived, at least

from a practical healthcare perspective, came in an article celebrating the joy of having *someone else* update the EHR:

> "Without much fanfare or planning, scribes have entered the scene in hundreds of clinics and emergency rooms. Physicians who use them say they feel liberated from the constant note-taking that modern electronic health records systems demand. Indeed, many of those doctors say that scribes have helped restore joy in the practice of medicine, which has been transformed — for good and for bad — by digital record-keeping."[1]

If you introduce a new IT system to help a particular professional do a better job, and one of the most celebrated advances in the use of that IT system is having someone else manage it so you don't have to come into contact with it, your system is pretty much condemned as nothing but a drag on the user. It's hard to think up a more damning verdict than that.

[1] The New York Times, January 14, 2014, "A busy doctor's right hand, ever ready to type", http://www.nytimes.com/2014/01/14/health/a-busy-doctors-right-hand-ever-ready-to-type.html

Chapter 9

Current EHR: Cosmic

I'll illustrate how current systems work by using examples from Cambio Cosmic, a system I've used quite a bit. The descriptions and screenshots aren't of the most recent version and some improvements have been made since I took these[1], but fundamentally, it's still working according to the same principles. I also want to stress that Cosmic is certainly not the worst system out there; I wouldn't be surprised at all if it was the best. At least, I can't claim I've seen anything significantly better out there. But it is definitely representative of the sorry state of systems we have.

In the first screenshot (figure 9.1) we see a window showing the notes for the current patient. At the top of the screen we have the demographics, showing the personal number and name of the patient (a fake test patient). The top right shows three buttons in different colors indicating different kinds of warnings. The right large pane shows the contents of notes selected from the left pane.

The left pane lets you select what to show in the right pane. Basically, it's a list of sources of documentation, typically different departments within primary care or hospital care. It's basically an organizational chart. County owned primary care is one section, while privately owned primary care centers have their own sections. Each speciality department, such as "urology" and "orthopedics", is grouped under a larger umbrella (such as "surgery"). This must be wonderful for a civil servant to see, but is pretty much pointless to the work of a doctor. Yes, we *do* want to know occasionally *where* the patient has been, but generally we're much more interested in *what* and *why*, which this list doesn't help with at all. Also, even though this list on the left is clearly intended for management, they're not really allowed to see it, since they're not involved in patient care and shouldn't

[1] I would like to use more recent screenshots, but I very much doubt they'd let me take them.

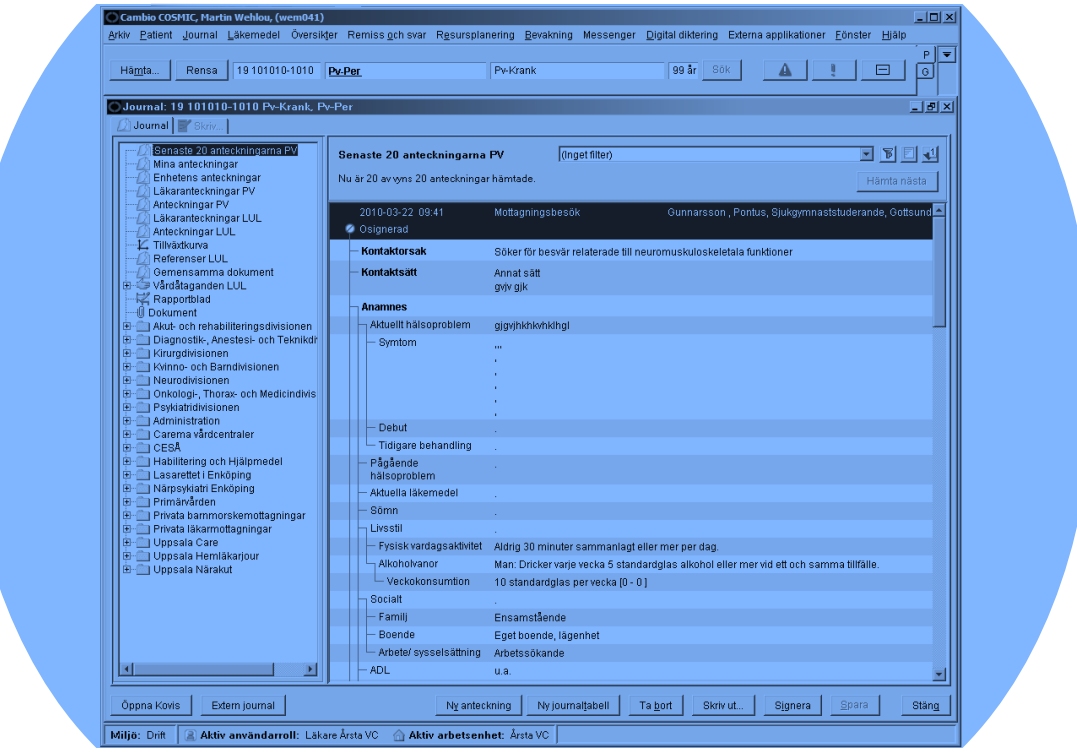


Figure 9.1: Cosmic, viewing the notes in the record

have this view of the record available.

This left pane is a fantastic illustration of what's wrong with current systems. It's pretty clear it was designed, or at least specified, by a lay person *imagining* what he or she would like to see there if he was a doctor, and then forcing doctors to see just that. But I very much doubt that a properly informed and independent doctor would have chosen that view to form such a dominating part of the workflow.

If you look at the top of the left pane in figure 9.1, you'll see two tabs. The left one says "Journal" ("Notes"), while the right one says "Skriv" ("Write"). This is where you switch both panes over from reading to writing.

When in writing mode, the left pane changes to show all the "keywords" or subtitles, you can enter information into (see figure 9.2). Whatever you write in the right pane is entered into the field contents of the keyword you have selected on the left. At the top, above the entry field in the right pane, you can select date, time, contact (encounter), and where you are. Again,

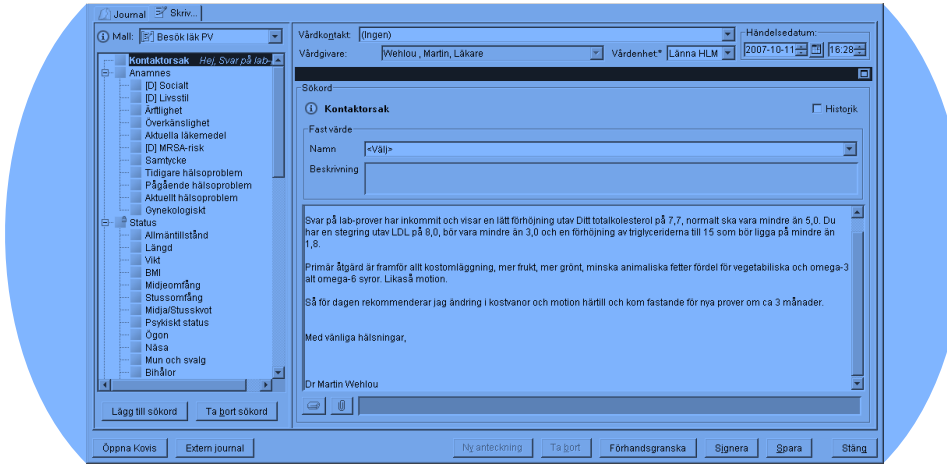


Figure 9.2: Cosmic in notes writing mode

in this screen, you see an emphasis on "who", "where", and "when", but not so much "what" and "why".

What we also see is a structure determined by those keywords to the left. This neatly shows what the software engineers have understood when we say "structured records", namely a text divided into sections according to some fairly random list of discernible terms. That's not what most doctors would regard as a medically sensible structure. Anyone can slice any text any number of ways without improving its utility. The "structure" we'd like to see is a division into diagnostic plans, intentions, deductions, findings, and the underpinnings of those. There's nothing of that in Cosmic, or in any other current EHR systems.

When creating a prescription in Cosmic, it looks like in figure 9.3. Clearly, the entire working space on the screen is occupied by all the stuff you need to prescribe a medication. The only other things that still appear are my name, the patient's name, and the contact date and time. That's it. It's obvious that the software engineers didn't see that prescriptions have anything to do with notes, referrals, lab, or just about anything else in the records. Or that the doctor would need to refer to anything else while creating prescriptions. When seeing a patient for hypertension, for instance, it's as easy (or difficult) to prescribe a blood pressure lowering medication as it is to prescribe the pill, or morphine, or a drug against Parkinson's disease. The system is totally oblivious to what you're doing. It doesn't help and it doesn't hinder (too much). If I send a referral, likewise the system allows me, with exactly the same level of support or hindrance, to

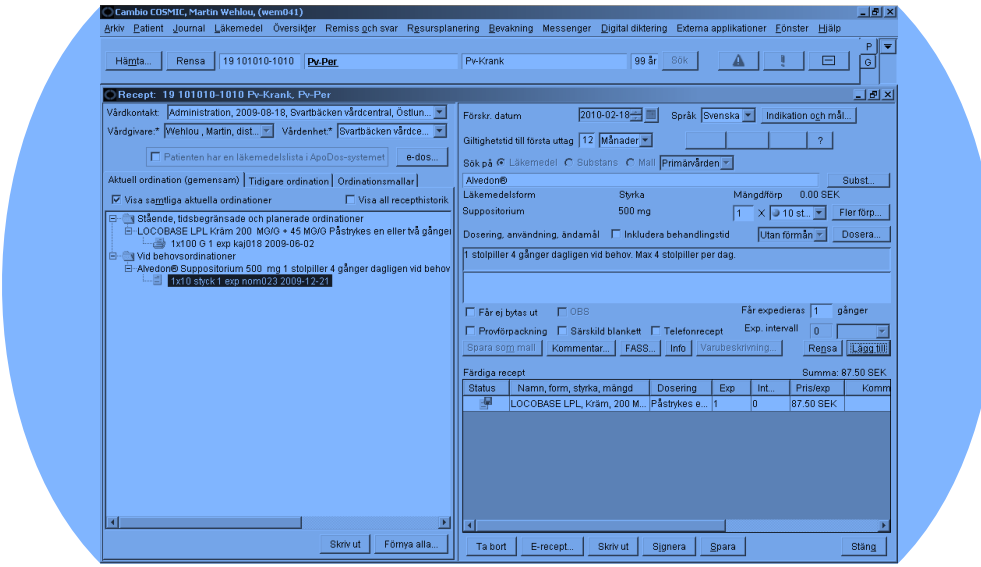


Figure 9.3: Creating a prescription in Cosmic

send it to a cardiologist, a kidney specialist, or, why not, a psychiatrist, or a hearing aid service. It has no preferences.

If you had created a system for photographers with this attitude and domain implementation, it would contain functions for finding models, taking pictures, developing film, making copies, and invoicing clients, flushing toilets, and clearing gutters, but the picture taking would have no relationship to either models or film, the development task would involve bottles of solutions, but no film or copies, while the invoicing would have no relationship to anything else, making it as easy to invoice a photo shoot as a dozen dead rabbits. No photographer in his right mind would buy an application like that.

I have no screenshot of how you write referrals with Cosmic, but you're not missing much. It's the same kind of thing, where once you start writing, you see nothing else, and cannot look up anything else. The referral itself is not connected to any particular disease or other documents, and is, for all practical purposes, as integrated into the patient's narrative as a dozen dead rabbits.

I said it in the beginning of this chapter and I'll say it again: *Cosmic isn't worse than anything else out there, it may even be the best system of them all. It's probably pretty much representative of the current market.* Which, when you think of it, isn't a very comforting thought.

Chapter 10

Knowledge support

As doctors or nurses, we can't possibly keep everything we should know just in our heads. And even if we could, things change as science advances. There is such an enormous amount of data we have to make use of, and an ever growing avalanche of new findings. There's no way we can drink from this fire-hose and do real work at the same time.

Depending on how fresh the information is, and the level of detail, it is available as original articles, review articles, textbooks, and/or guidelines.

10.1 Original articles

According to the Medline Fact Sheet[1], there are more than 19 million references to original articles in their database. Another 2,000 - 4,000 are added *daily*. Even though everything we need to know in our daily practice is in there *somewhere*[2], this information isn't useful in clinical practice in its raw form. Researchers working on a particular topic have great use of this resource, but for a clinician having a patient in her office, it's practically useless. You can't just say to the patient: "I have to go read up on your problem for a bit. Grab a coffee, and I'll be back in a couple of weeks."

At times, publications containing remarkable and important discoveries are widely read by doctors, but this is just an infinitesimally small fraction of the total published mountain of articles. And we can't just ignore the rest. Most of them form the basis of future diagnostic and therapeutic principles, so we have to use a system to reduce them to a more digestible form.

[1] http://www.nlm.nih.gov/pubs/factsheets/medline.html

[2] I'm glossing over the practical techniques, the judgement, and the experience we need as doctors, focusing solely on the scientific findings.

10.2 Review articles

Well-informed authors regularly write reviews of the most important papers in their field and publish those in the same medical journals where the original papers are usually published. These review articles allow non-specialist doctors to get a good overview of a particular subject and the current state of knowledge about it. While writing these reviews, the authors use their best judgement to sift through the underlying original articles to bring forward the most relevant and trustworthy findings, so the rest of us don't have to do that.

These review articles are still too detailed and specialized to be directly useful in patient care in general, but are often just fine as a tool to keep abreast of an area where the doctor sees a lot of patients, but not necessarily does research.

10.3 Textbooks

Textbooks constitute the next level up, and the quality of the information is similar to reviews, but as a collection of related subjects for a certain medical speciality.

While original articles and review articles presume the reader is already familiar with the subject, the textbook does not, and starts from basics. Textbooks are rarely useful for the practicing doctor, except as a reminder of what he or she learned in medical school. Additionally, text books are much too expensive to buy just to keep up with the science. They're also not updated quickly enough for that use.

10.4 Guidelines

Guidelines are based on original research and reviews, and turn that content into practical use advice. If there's a study that says it's advantageous to the patient to apply therapy X, the guideline recommends therapy X, at least if it's available in the relevant region.

In theory, guidelines are entirely based on science, the interest of the patient, and the resources available, and are short and sweet enough that they can be read, understood, and applied while the patient is present. In practice, however, these guidelines are often influenced by considerations that have nothing to do with the patient's immediate interest, but by political and economic incentives. If those economic considerations are of the kind that optimizes healthcare in a larger perspective, it's reasonable to accept that, but manipulative bureaucrats sometimes compromise the integrity of these motives to such a degree that guidelines are getting a bad rap as

being a tool for civil servants to control medical professionals, rather than a tool to improve the quality of the direct patient work. If we can't trust the motives behind a guideline, we're not likely to seek it out and use it, either.

Somewhat unexpectedly, it's hard to find useful guidelines in English, but one decent example can be seen at a site loosely connected to Oxford University (according to the text on the site). The guideline I looked at for the purposes of this discussion, is their guideline on the work-up and management of chronic heart failure, and I found it at this URL[3], which, of course, has a good chance of not existing anymore once you try it. That's the internet for you.

This guideline is in the form of a PDF file and summarizes the causes, the diagnostic procedures, and the recommended treatments stratified into levels of seriousness. The guideline document includes text, flowcharts, tables, and forms. Even though the information is solid and useful, the document is a mashup of different kinds of functionality, all squeezed into one single document, severely impacting its practical usefulness. Let's look at a few aspects of this guideline and its construction, and it'll become clearer just what I mean by that.

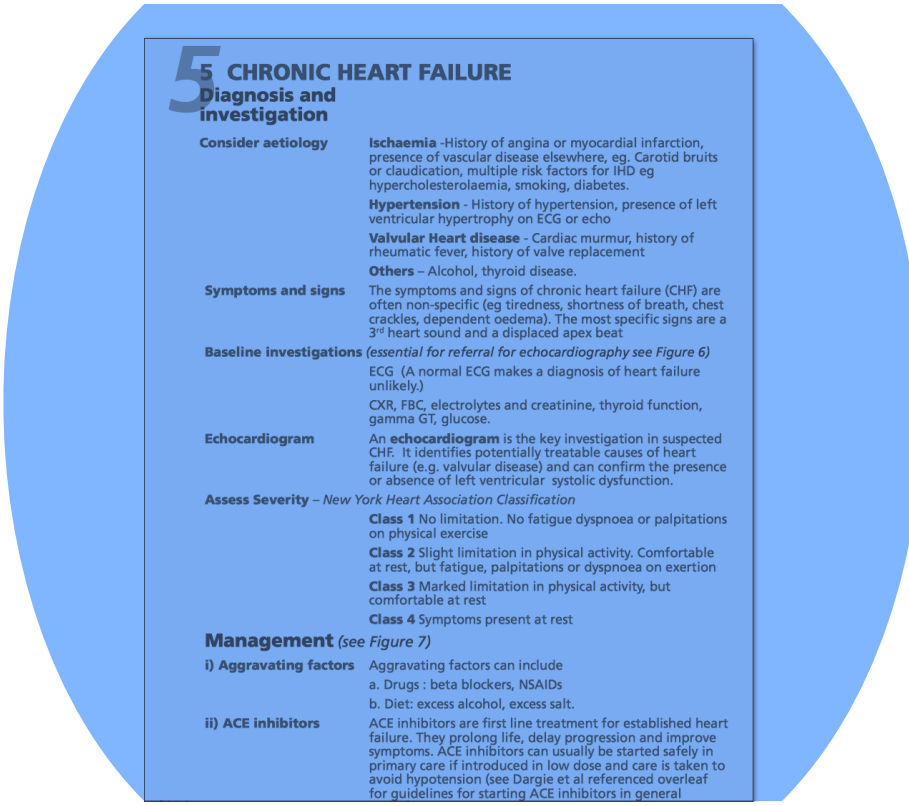
In figure 10.1 we see the beginning of the guideline, where there is a short overview of the basic diagnostic procedures. As doctors we're supposed to know all this by heart, but we don't always remember every detail. It could be too long ago, too detailed, or we can have a bad day, so this is excellent as a brief reminder.

Figure 10.2 shows a part of the therapeutic discussion from the guideline. Again, it's terrific as a reminder, but it is so much more than that. It's also an *up to date* reminder, (or it *should* be[4]), potentially alerting us to changes in recommended therapies caused by new discoveries that may have shown newer agents to be more effective than older agents, or shown some agents not to be as effective or free of adverse effects as we thought. In other words, however well you know past recommendations by heart, checking through this list can always teach you something new and bring you up to date.

In figure 10.3 we see more or less the same information, but in tabular form, with indications of trustworthiness in the "evidence" column, and with literature references in the last column. That last column is essential, since it ties the recommendations to the underlying original data. The guideline should never be the expression of just the author's personal preferences, it must *provably* be based on neutral scientific evidence, and that's where this

[3] http://static.oxfordradcliffe.net/med/gems/CHF.pdf

[4] It's absolutely essential that the guideline includes the date it was last updated, so that we can judge its actuality, but this guideline does not. That is a serious shortcoming.



5 CHRONIC HEART FAILURE
Diagnosis and investigation

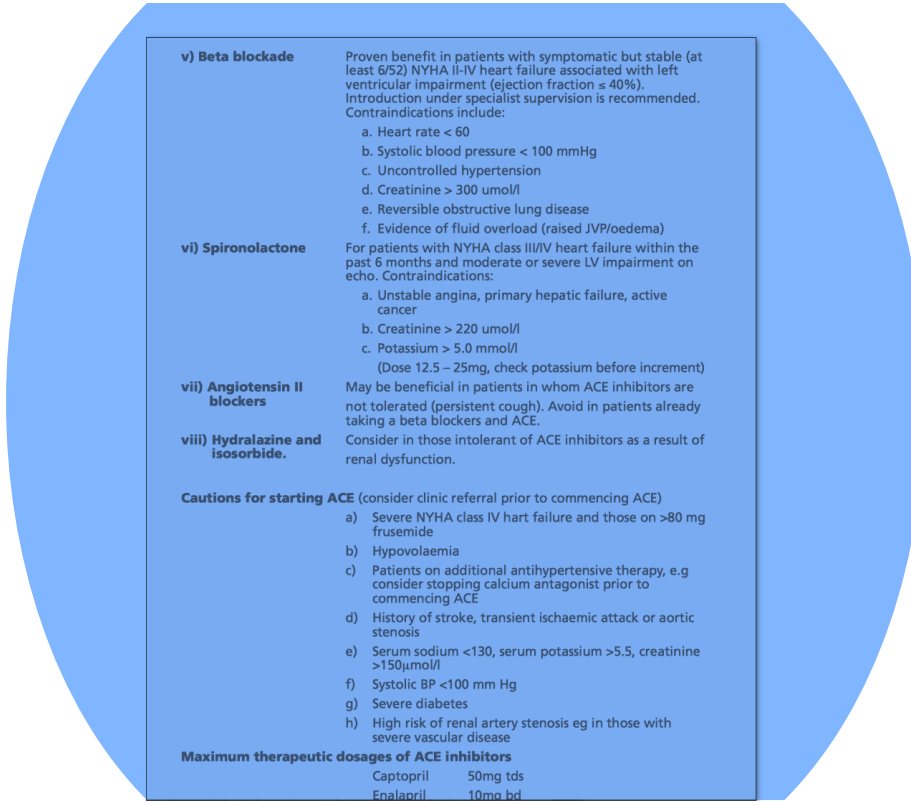
Consider aetiology	**Ischaemia** -History of angina or myocardial infarction, presence of vascular disease elsewhere, eg. Carotid bruits or claudication, multiple risk factors for IHD eg hypercholesterolaemia, smoking, diabetes.
	Hypertension - History of hypertension, presence of left ventricular hypertrophy on ECG or echo
	Valvular Heart disease - Cardiac murmur, history of rheumatic fever, history of valve replacement
	Others – Alcohol, thyroid disease.
Symptoms and signs	The symptoms and signs of chronic heart failure (CHF) are often non-specific (eg tiredness, shortness of breath, chest crackles, dependent oedema). The most specific signs are a 3rd heart sound and a displaced apex beat
Baseline investigations	*(essential for referral for echocardiography see Figure 6)*
	ECG (A normal ECG makes a diagnosis of heart failure unlikely.)
	CXR, FBC, electrolytes and creatinine, thyroid function, gamma GT, glucose.
Echocardiogram	An **echocardiogram** is the key investigation in suspected CHF. It identifies potentially treatable causes of heart failure (e.g. valvular disease) and can confirm the presence or absence of left ventricular systolic dysfunction.
Assess Severity – *New York Heart Association Classification*	
	Class 1 No limitation. No fatigue dyspnoea or palpitations on physical exercise
	Class 2 Slight limitation in physical activity. Comfortable at rest, but fatigue, palpitations or dyspnoea on exertion
	Class 3 Marked limitation in physical activity, but comfortable at rest
	Class 4 Symptoms present at rest
Management *(see Figure 7)*	
i) Aggravating factors	Aggravating factors can include
	a. Drugs : beta blockers, NSAIDs
	b. Diet: excess alcohol, excess salt.
ii) ACE inhibitors	ACE inhibitors are first line treatment for established heart failure. They prolong life, delay progression and improve symptoms. ACE inhibitors can usually be started safely in primary care if introduced in low dose and care is taken to avoid hypotension (see Dargie et al referenced overleaf for guidelines for starting ACE inhibitors in general

Figure 10.1: Guideline for management of chronic cardiac failure, diagnosis and investigation.

last column comes in. Without an explicit base in original scientific work, the guideline loses most of its value.

Not everything in a guideline can be based on explicit scientific material, however. Some of it is based on generally agreed good practices, but it's important that those parts of the advice in the guideline are clearly recognized as such. The reader should always have the ultimate say in how much authority he or she lends to the guideline in the individual patient case, and should not have to take anything in the guideline on faith, or as a result of belief in the authority of the author alone.

When we're working through a guideline, in particular the recommendation parts, we always weight the recommendations against the totality of the patient we have in front of us. Some recommendations can't be followed due to other diseases the patient has, personal preferences of the patient,



v) Beta blockade — Proven benefit in patients with symptomatic but stable (at least 6/52) NYHA II-IV heart failure associated with left ventricular impairment (ejection fraction ≤ 40%). Introduction under specialist supervision is recommended. Contraindications include:

a. Heart rate < 60
b. Systolic blood pressure < 100 mmHg
c. Uncontrolled hypertension
d. Creatinine > 300 umol/l
e. Reversible obstructive lung disease
f. Evidence of fluid overload (raised JVP/oedema)

vi) Spironolactone — For patients with NYHA class III/IV heart failure within the past 6 months and moderate or severe LV impairment on echo. Contraindications:

a. Unstable angina, primary hepatic failure, active cancer
b. Creatinine > 220 umol/l
c. Potassium > 5.0 mmol/l

(Dose 12.5 – 25mg, check potassium before increment)

vii) Angiotensin II blockers — May be beneficial in patients in whom ACE inhibitors are not tolerated (persistent cough). Avoid in patients already taking a beta blockers and ACE.

viii) Hydralazine and isosorbide. — Consider in those intolerant of ACE inhibitors as a result of renal dysfunction.

Cautions for starting ACE (consider clinic referral prior to commencing ACE)

a) Severe NYHA class IV hart failure and those on >80 mg frusemide
b) Hypovolaemia
c) Patients on additional antihypertensive therapy, e.g consider stopping calcium antagonist prior to commencing ACE
d) History of stroke, transient ischaemic attack or aortic stenosis
e) Serum sodium <130, serum potassium >5.5, creatinine >150μmol/l
f) Systolic BP <100 mm Hg
g) Severe diabetes
h) High risk of renal artery stenosis eg in those with severe vascular disease

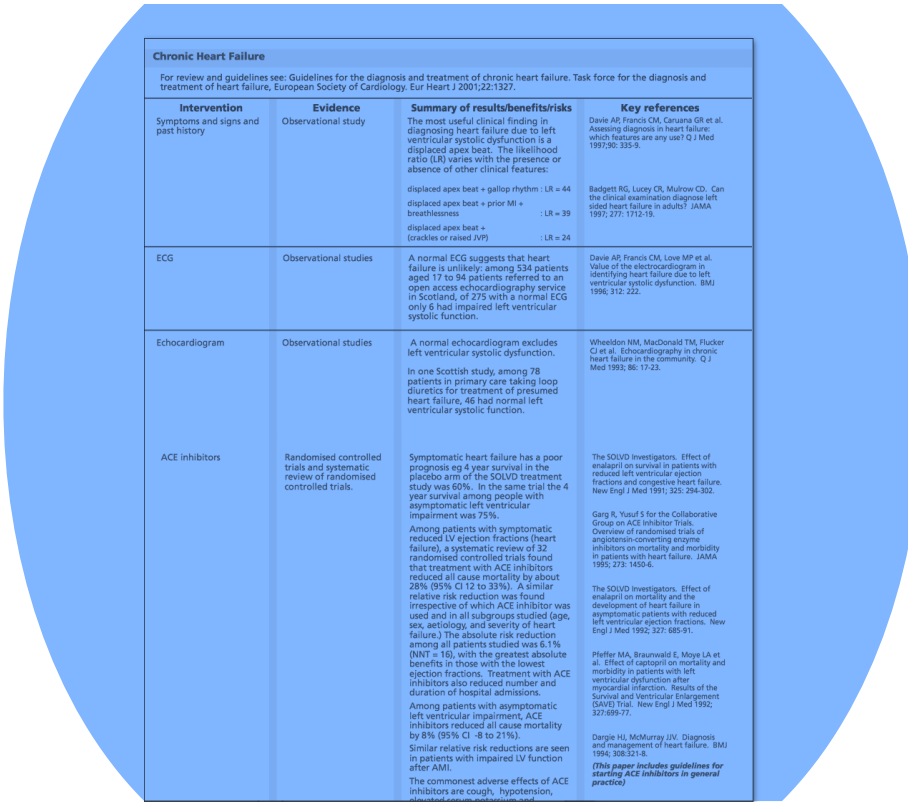
Maximum therapeutic dosages of ACE inhibitors

Captopril	50mg tds
Enalapril	10mg bd

Figure 10.2: Chronic cardiac failure, part of the treatment description.

and many more factors. It's impossible, and unproductive, to even attempt to include all those considerations into a guideline. If we as doctors weren't able to take those considerations into account ourselves, we shouldn't be doing this job, anyway.

But here's a problem. Reading the guideline and selectively applying the advice based on considerations that are not part of the text leaves no trace in the medical record. While I'm reviewing the guideline and reacting to it, I'm performing the real essence of my profession, and I'm taking the important decisions about the patient's care, but it's not persisted anywhere. I can't scribble on the screen, striking through stuff, or underlining, adding checkmarks, etc. Yes, I *can*, but the IT department won't appreciate it. The guideline is my worksheet, the very embodiment of my reasoning, and I can't save it? That's ridiculous! In other words, however great the content of the guideline is, the form is wrong. It shouldn't be presented as



Chronic Heart Failure

For review and guidelines see: Guidelines for the diagnosis and treatment of chronic heart failure. Task force for the diagnosis and treatment of heart failure, European Society of Cardiology. Eur Heart J 2001;22:1327.

Intervention	Evidence	Summary of results/benefits/risks	Key references
Symptoms and signs and past history	Observational study	The most useful clinical finding in diagnosing heart failure due to left ventricular systolic dysfunction is a displaced apex beat. The likelihood ratio (LR) varies with the presence or absence of other clinical features: displaced apex beat + gallop rhythm : LR = 44 displaced apex beat + prior MI + breathlessness : LR = 39 displaced apex beat + (crackles or raised JVP) : LR = 24	Davie AP, Francis CM, Caruana GR et al. Assessing diagnosis in heart failure: which features are any use? Q J Med 1997;90: 335-9. Badgett RG, Lucey CR, Mulrow CD. Can the clinical examination diagnose left sided heart failure in adults? JAMA 1997; 277: 1712-19.
ECG	Observational studies	A normal ECG suggests that heart failure is unlikely: among 534 patients aged 17 to 94 patients referred to an open access echocardiography service in Scotland, of 275 with a normal ECG only 6 had impaired left ventricular systolic function.	Davie AP, Francis CM, Love MP et al. Value of the electrocardiogram in identifying heart failure due to left ventricular sytolic dysfunction. BMJ 1996; 312: 222.
Echocardiogram	Observational studies	A normal echocardiogram excludes left ventricular systolic dysfunction. In one Scottish study, among 78 patients in primary care taking loop diuretics for treatment of presumed heart failure, 46 had normal left ventricular systolic function.	Wheeldon NM, MacDonald TM, Flucker CJ et al. Echocardiography in chronic heart failure in the community. Q J Med 1993; 86: 17-23.
ACE inhibitors	Randomised controlled trials and systematic review of randomised controlled trials.	Symptomatic heart failure has a poor prognosis eg 4 year survival in the placebo arm of the SOLVD treatment study was 60%. In the same trial the 4 year survival among people with asymptomatic left ventricular impairment was 75%. Among patients with symptomatic reduced LV ejection fractions (heart failure), a systematic review of 32 randomised controlled trials found that treatment with ACE inhibitors reduced all cause mortality by about 28% (95% CI 12 to 33%). A similar relative risk reduction was found irrespective of which ACE inhibitor was used and in all subgroups studied (age, sex, aetiology, and severity of heart failure.) The absolute risk reduction among all patients studied was 6.1% (NNT = 16), with the greatest absolute benefits in those with the lowest ejection fractions. Treatment with ACE inhibitors also reduced number and duration of hospital admissions. Among patients with asymptomatic left ventricular impairment, ACE inhibitors reduced all cause mortality by 8% (95% CI -8 to 21%). Similar relative risk reductions are seen in patients with impaired LV function after AMI. The commonest adverse effects of ACE inhibitors are cough, hypotension,	The SOLVD Investigators. Effect of enalapril on survival in patients with reduced left ventricular ejection fractions and congestive heart failure. New Engl J Med 1991; 325: 294-302. Garg R, Yusuf S for the Collaborative Group on ACE Inhibitor Trials. Overview of randomised trials of angiotensin-converting enzyme inhibitors on mortality and morbidity in patients with heart failure. JAMA 1995; 273: 1450-6. The SOLVD Investigators. Effect of enalapril on mortality and the development of heart failure in asymptomatic patients with reduced left ventricular ejection fractions. New Engl J Med 1992; 327: 685-91. Pfeffer MA, Braunwald E, Moye LA et al. Effect of captopril on mortality and morbidity in patients with left ventricular dysfunction after myocardial infarction. Results of the Survival and Ventricular Enlargement (SAVE) Trial. New Engl J Med 1992; 327:699-77. Dargie HJ, McMurray JJV. Diagnosis and management of heart failure. BMJ 1994; 308:321-8. ***(This paper includes guidelines for starting ACE inhibitors in general practice)***

Figure 10.3: Summary table with literature references for chronic cardiac failure.

a read-only webpage or even paper document, unless I can scribble on it, modify it, and make it part of the patient's ongoing record.

The form we see in figure 10.4 only underlines this fatal flaw. The form is just sitting there, totally passive. Yes, I can print it out and fill it out by hand[5]. The idea is fine, providing us with the right form, but the delivery is deeply flawed. This form should be different according to the organization and geographical location of the user, be sent automatically to the destination, become part of the patient's record automatically, and become connected to the reply, once it arrives. Clearly, this is too much

[5]Since the death of the typewriter, there's no other way to fill in forms, really. Unless you count spending hours installing and *cursing* Adobe Reader to kingdom come, for the pleasure of *trying* to hit those fields, then *going ballistic* as it can't save the filled in document anywhere, least of all in the medical record.

HEART FAILURE ECHO CLINIC REFERRAL FORM

In order to supply an adequate report and advice please provide as much of the information below as possible

NHS

Date of referral / /
Patient name:
Second name
D.O.B
Address
New NHS No.
Daytime Tel:
GP Name:
Address
Fax No.
Phone No.

Automatic referral to outpatient clinic – review by consultant team in outpatients? Yes No
(Pease tick ✓ if required)

History:

Drugs:

Cardiac history: *Please indicate*

Hypertension Yes No | Diabetes Yes No | Hypercholesterolaemia Yes No
Valve disease Yes No | Smoking Yes No | Family history Yes No
MI Yes No | Angina Yes No

Investigation results (Abnormal only):
FBC | U+E | Liver function | TSH | Creatinine | Gamma GT | Glucose | Cholesterol

Please attach or forward CXR report and ECG (compulsory)

Signed: Name (PRINT):

Figure 10.4: Referral form for chronic cardiac failure.

too ask of a plain dumb PDF file on a website.

In figure 10.5, we see the final page of the guideline, with a neat flowchart representation of the management of chronic heart failure. This representation is great for a review of the guideline by the user, but since it's entirely passive, it has no immediate use in any software implementation.

I have a more philosophical disagreement with the flowchart representation, namely that it invites to being implemented as a process in medical records, and that would be tragic for two reasons. First, we have the "keyhole" effect I'm discussing on page 185, that is an incentive for the user to hunt around in the program, answering questions in different ways just to see what conclusions and recommendations come out of the software. The other reason is that it is an invitation to lay management to try to automate away the need for doctors. The flowchart, after all, seems to indicate that the entire decision process can be reduced to just a few input factors

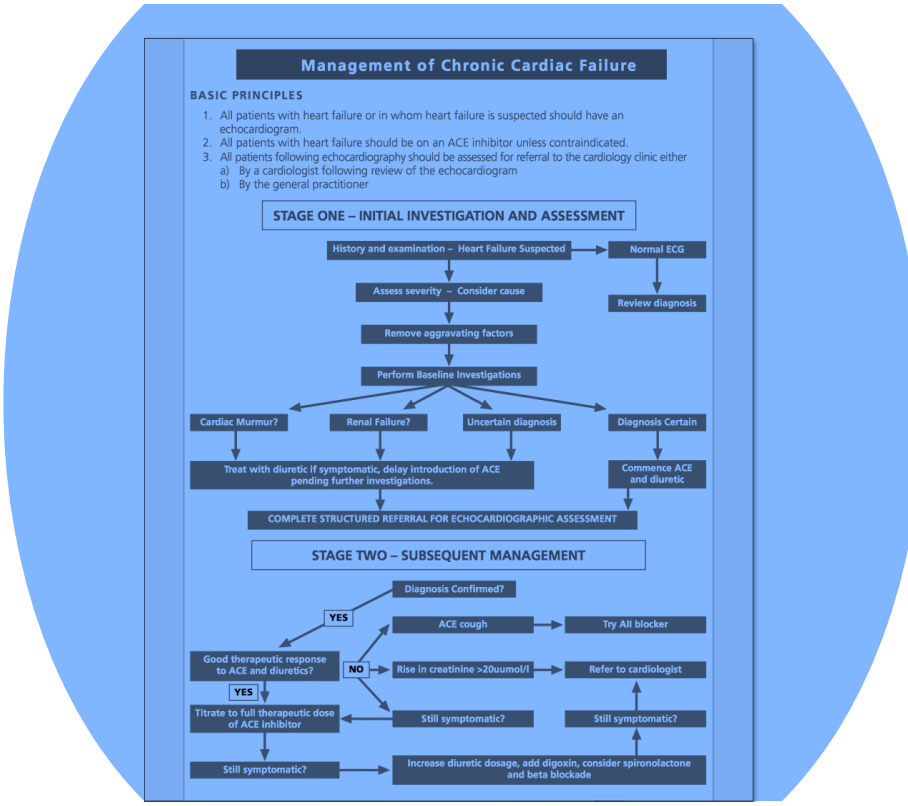


Figure 10.5: Flowcharts for investigation and management of chronic cardiac failure.

and a few decision branches. Lay people tend not to see what we're using our training and experience for, once it's ostensibly reduced to a flowchart. In short, I think the flowchart here has more potential for damage than for good use.

10.5 Continued Professional Education

In most countries[6], doctors must follow a minimum amount of continued professional education per year. The idea is to bring us up to speed, at least in our own speciality. This idea falls down when scrutinized, though.

[6]Sweden has no such obligation. I don't know what to think of that, really.

Let's take a general practitioner as an example. We have a few hundred major diseases we diagnose and treat with some regularity. If we assume we can work through at most five of those per year, it would take us half a century or more to get through them all and wrap around to the beginning. Since the mean length of a GP career is around 35 years, an individual doctor would never get continued education even once on more than half of these most frequent diseases.

Also consider how much you can possibly remember from an education session, and for how long. Personally, I think the information is at the top of your brain, and in a useful state, for a couple of weeks at the most. After that, it sinks down into the same swamp all the other knowledge from university days bubbles around in, and becomes effectively almost useless. Any patients with that disease you see while still having the information fresh in your mind will benefit, and possibly make the information more long lasting. But how many Parkinson patients will you see in the three first weeks after that one session on Parkinson you had last year? One?

I'm not entirely alone in this pessimism. A paper from the Wellcome Trust[7] says:

> "The researchers estimate that the time lag between research expenditure and eventual health benefits is around 17 years."

That number, 17 years, is pretty much the midpoint of the average career for a doctor, leading me to my own informal conclusion:

> *Nobody learns much of anything after graduating from medical school. Continued professional education, in practice, makes no useful difference.*

Of course, there are exceptions to my very pessimistic rule. Any training in manual dexterity, what I call "craftsmanship" on page 4.4, is best taught using in-person meet ups at nice hotels or spas with excellent restaurants and bars. But for theoretical knowledge, I very much doubt that this is an effective method.

[7] http://www.wellcome.ac.uk/stellent/groups/corporatesite/@sitestudioobjects/documents/web_document/wtx052110.pdf

Chapter 11

How is the record created?

Before designing an interface, and even before analyzing the meaning of the entered data, we have to know how and when the user creates the input. It is the physical environment and context around the place and time of input that largely decides the character and value of the data produced.

11.1 The input method

There are several ways data can be entered into the medical record. There are three major workflows, each with their own advantages and disadvantages.

Writing afterwards

Some doctors take short handwritten notes during the encounter, then transcribe those into long form notes in the medical record system after the patient leaves. This implies that the doctor has some time alone between patients, or that he/she spends time at the end of the day transcribing notes for several patients.

Usually, referrals, reports, and attestations of different kinds are written in that same period, after the patient leaves. Some doctors write referrals and attestations with the patient present, but leave the dictation or writing of the record for after the patient has left.

Dictating afterwards

In this workflow, the doctor also takes some brief notes during the encounter, then arranges his thoughts and proceeds to dictate the notes after the patient has left, or possibly after hours. It's also easier to mix up the

history and findings of different patients if the dictation is all done at once like this.

Simultaneous writing

In this workflow, the doctor takes notes during the actual encounter, as the information is provided and develops.

This way of working is surprisingly easy on doctor and patient, at least I think so. As the doctor reacts visibly by typing to almost everything the patient says, the patient feels he or she being taken seriously, is less prone to repetition, and also becomes more succinct and to the point. That's all good.

The major obstacle to this is that the doctor needs to learn touch-typing, to be able to maintain eye contact with the patient, at least part of the time. But that isn't hard to achieve with a little training. It's orders of magnitude easier to learn to type well, as compared to learning to become a doctor.

Another advantage of writing while the patient is present is that if any point is missing or unclear in the narrative, it is much more straightforward to get clarification of these points as the patient is still there to help out.

Finally, since all note taking and writing of referrals is done before the patient leaves, there is no work left to do in the interval between patients or after hours. Also, the patient gets to be with the doctor for a longer period than if the writing or dictating needs to be done in his absence.

Interestingly, in some countries doctors almost always write the record themselves, while in other countries it's rare. It clearly is a cultural phenomenon.

Obstacles to simultaneous writing

The major obstacle is the lack of typing ability in doctors, and even worse, their unwillingness to learn. This can and should be overcome with suitable incentives.

Another more serious obstacle is the poor design of most EHR systems today. In most of them, it is impossible to refer to all the source documents you need while creating typical output. Often the editor window you use to write notes is the same window you use to look up older notes. The same happens in referrals; while writing a referral you can't go back and read another referral, since these functions utilize the same window. You can't save a referral as draft to go look something up, so you need to throw the draft away, go look up something, then start over. Adding insult to injury, you can't reuse an earlier referral, modify it and send it again. Since one of the most frequent workflows you need to handle is re-composing a referral

that was bounced back from a referee[1], these defects combine to a perfect storm of aggravation.

The systems are much easier to use if you *either* read documents *or* write them, but not both at the same time. It is also often extremely complicated to copy existing information into new forms and notes. Copy and paste is often poorly implemented, and even if it works, it's a very cumbersome method. If you dictate your records, this is not a problem, since the doctor then reads, while the secretary writes, which explains this terrible design. The systems are in effect designed for a workaround; having a secretary type the medical record is a workaround for bad designs and defective medical records, and the medical records vendors keep designing for that workaround, perpetuating the problem.

The way the EHR systems are designed needs to be changed to take simultaneous writing into account. This, again, is probably different ~~according~~ to the culture of the country we're in.

11.2 The different results

If the notes are created *after* the actual encounter with the patient, the contents of the notes will be different from those that could have been written *during* an encounter. The reason for this is that in the first case, the notes are an after-construction, where findings are already filtered through the conclusion the doctor reached. It's, in other words, colored by prejudice. As an example, we'll take the following scenario. If the doctor does listen to the patient telling him about his sore right knee in full detail, and the patient also mentions he had a rash and a fever two weeks before his knee acted up, the rash and the fever will probably not end up in the patient history notes that the doctor writes down, unless he sees a connection between that and the sore knee, the principal complaint. There is no way for another doctor, or a computer, to diagnose any problem with the knee that also involves the rash and fever, by analyzing the notes only. The information needed to do so will simply not be there.

If, on the other hand, the doctor writes down the notes while the patient tells his story, he'll probably write down the part about the rash and the fever long before reaching a conclusion and a diagnosis that does not really include the fever and rash. As long as the doctor doesn't go through the trouble of going back and erasing that part of the history, intentionally screwing with the veracity of the record, the findings will indeed remain part of the record and may in the future lead to a different conclusion by another doctor or a machine, from the same data.

[1] Yes, a "referee" is someone in black and white stripes and a whistle, but it also means a person receiving a referral. I looked it up.

The greatly increased veracity and usefulness of the medical record in this last scenario, is a strong argument for having the doctor create the record *while* the patient is telling his story, and *during* the clinical examination, not afterwards.

> *It's of ultimate importance that the record is created as early in the process as possible, during the actual encounter. The only method that can be usefully employed when the patient is present is typing, or dictating, if the quality of the doctor-patient relationship allows it.*

11.3 How I am doing it

To illustrate, the following workflow the author himself employs while seeing patients.

1. I determine which patient is to be seen next, then I go out to fetch him or her.
2. After checking that I have the right patient, I open the patient's record and do a quick read. I also get the cause for the current consultation from the appointment listing and enter that into a new note in the EHR.
3. I ask the patient to tell me about his or her problem, while asking clarifying questions, typing in the history notes simultaneously.
4. I progress to the clinical examination, writing in the notes as I go.
5. I draw my conclusions, writing them down while I do so, create any referrals or prescriptions, sign off on everything, then escort the patient to the door.

The fact that I'm typing at the same time as the patient is telling his story is frowned upon by many in the medical field, on the presumption that the patient would want my undivided attention and not have me fiddle with a keyboard while he's talking, but this is utter nonsense. The patient quickly notices that whatever he says results in me writing it down, an obvious feedback loop, which demonstrates to the patient that I am indeed listening. By contrast, if I'm instead just calmly, but intensely, staring at the patient without moving, it's very hard for the patient to figure out if I'm listening or thinking about something else entirely, so he'll often repeat himself for emphasis or he goes off on tangents. Having me visibly react to almost every word really sharpens the patient and whittles down the history to the bare essentials, without undue prompting.

Writing all prescriptions and referrals while the patient is still present has several distinct advantages:

- If detailed information is needed for a referral, I can simply ask the patient for those details.
- Referrals depart as soon as possible, i.e. immediately, saving the patient unnecessary waiting.
- The patient observes me sending the referrals, which I usually punctuate with something like "so, that one is sent off already", making it very clear to the patient that if delays occur, they're not due to me being slow to send them off.

Since I'm doing all this while the patient is present, the contact time with the patient is longer, but the total time spent per patient is the same or shorter than with the more common dictation method. Also, I don't leave any patient administration cluttering up my desk or mind for when I see the next patient, both reducing my own mental load, and providing the next patient my undivided attention.

I simply can't understand why anyone would want to work any other way.

11.4 Where is the record created?

Current EHR systems are mysteriously designed to work on one particular computer in one particular place, the doctor's office, while doctors often have their office and examination rooms separate, or see patients during rounds. It's as if the people specifying or designing these systems have never actually seen a doctor at work. I'll discuss this in more detail when describing the clinical examination (section 23.3).

Chapter 12

The information model

So, how does the data model[1] look in the EHR, and what can we do to improve on it?

Legacy EHR systems generally provide a couple of major categories under which it sorts the documents belonging to a patient. These categories are typically:

- Daily notes.
- Referrals and replies to referrals.
- Radiology orders and protocols.
- Lab orders and results.
- Prescriptions.
- General correspondence.
- Forms of all kinds filled in for the patient.

Let's first look at how these legacy systems relate these categories to each other. Basically, their data model is has the patient at the top, and we then have a list of encounters, and another list of documents. If we're lucky, the EHR system relates referrals, prescriptions, notes, and other documents to the encounters where they were created, but in many systems even this is too much to ask.

Since the notes from one encounter can relate to more than one problem, while documents such as referrals or prescriptions relate to a particular

[1] "Data" is "information" in a form that a computer can process. "Information" is derived from the "data".

problem (there are exceptions), the relationship between encounters and documents, if it exists at all, is wrong and misleading. Additionally, documents often arrive outside the context of encounters, such as replies to referrals.

In fact, this organization into separate stacks show a frightening similarity to my Belgian predecessor's stacks I described on page 41. We don't seem to have progressed much beyond that, except we now have a set of stacks per patient, where there were only two stacks for the entire practice previously.

For the rest of this discussion, just to keep the diagrams simple, I'll limit myself to four stacks per patient, namely: notes, lab, x-ray, and referrals, as in figure 12.1. The first stack with the notes should really be represented as several stacks if we're talking about large unified systems, since each speciality has its own stack of notes, but still largely share the other stacks with lab, x-rays, referrals, etc, with each other. I'm almost starting to long back to simpler times when we had just two stacks in my Belgian practice. Yes, they were larger, but there were just the two, and they made a twisted kind of sense.

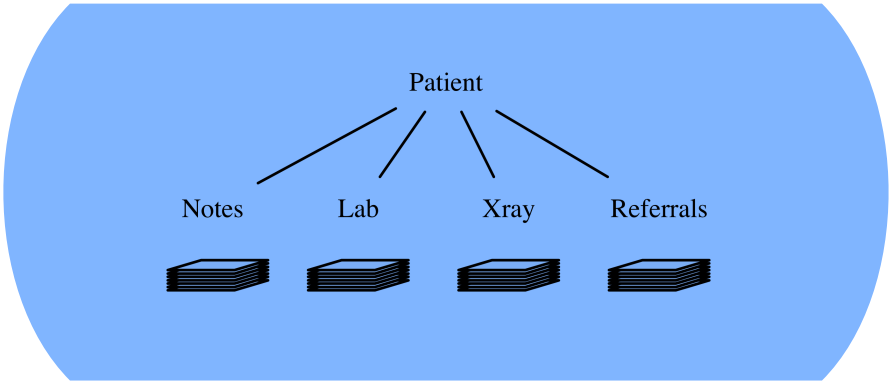


Figure 12.1: The "a few stacks" oriented architecture of legacy EHR systems

The legacy EHR systems have another dimension as well: time. Each of the stacks is organized in reverse time order, so the most recent documents or notes are uppermost (again, just like my Belgian predecessor did it). If we expand the stacks in figure 12.1 along the time axis, it will look something like figure 12.2.

There is nothing explicitly connecting any particular documents in one stack with related documents in another stack. In other words, if we as user of this system try to gather all the documents and notes that are relevant to a particular problem, it quickly degenerates into an error prone and

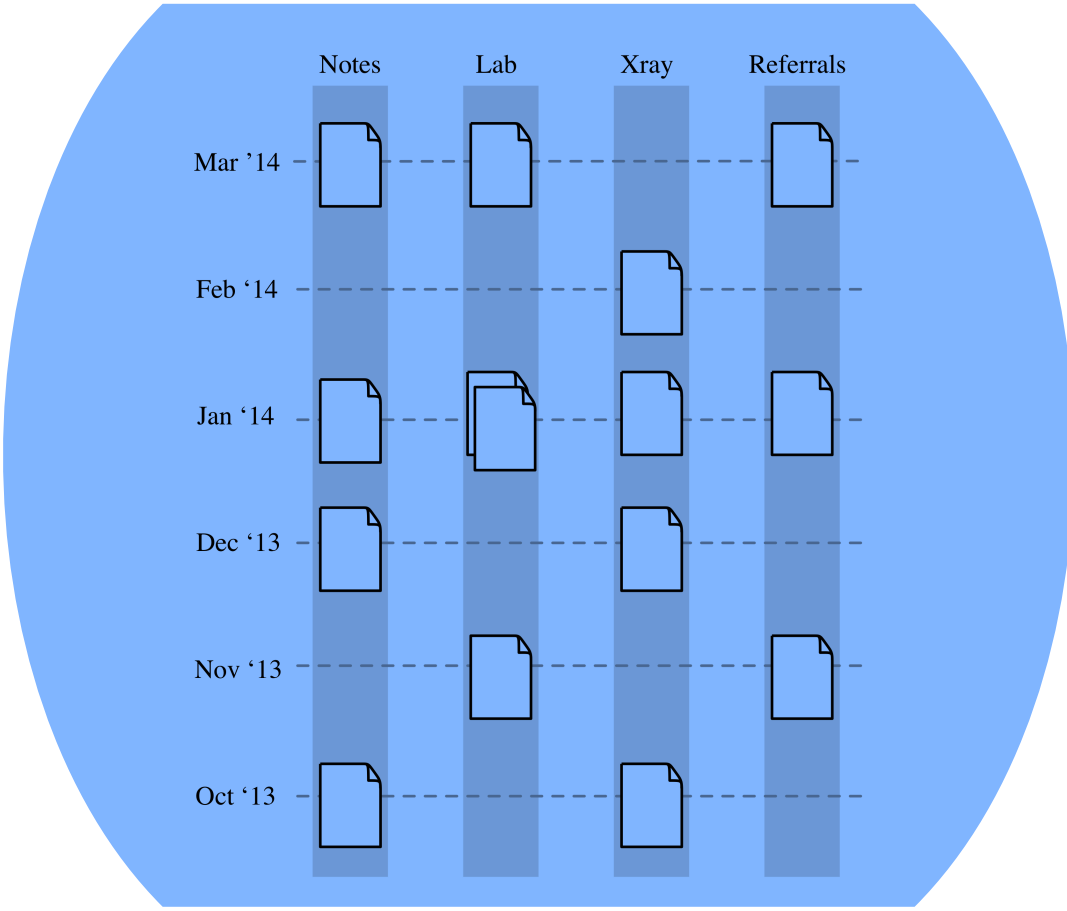


Figure 12.2: Legacy EHR document architecture

highly frustrating stack-walking procedure. Let's assume that the patient records in figure 12.2 contain information on three diseases, one of which is hypertension, for which we've done two lab orders, a chest x-ray, a renal x-ray, and two referrals (one for fundoscopy, the other to a vascular lab). In figure 12.3 I've marked the documents relating to hypertension with a crosshatched pattern.

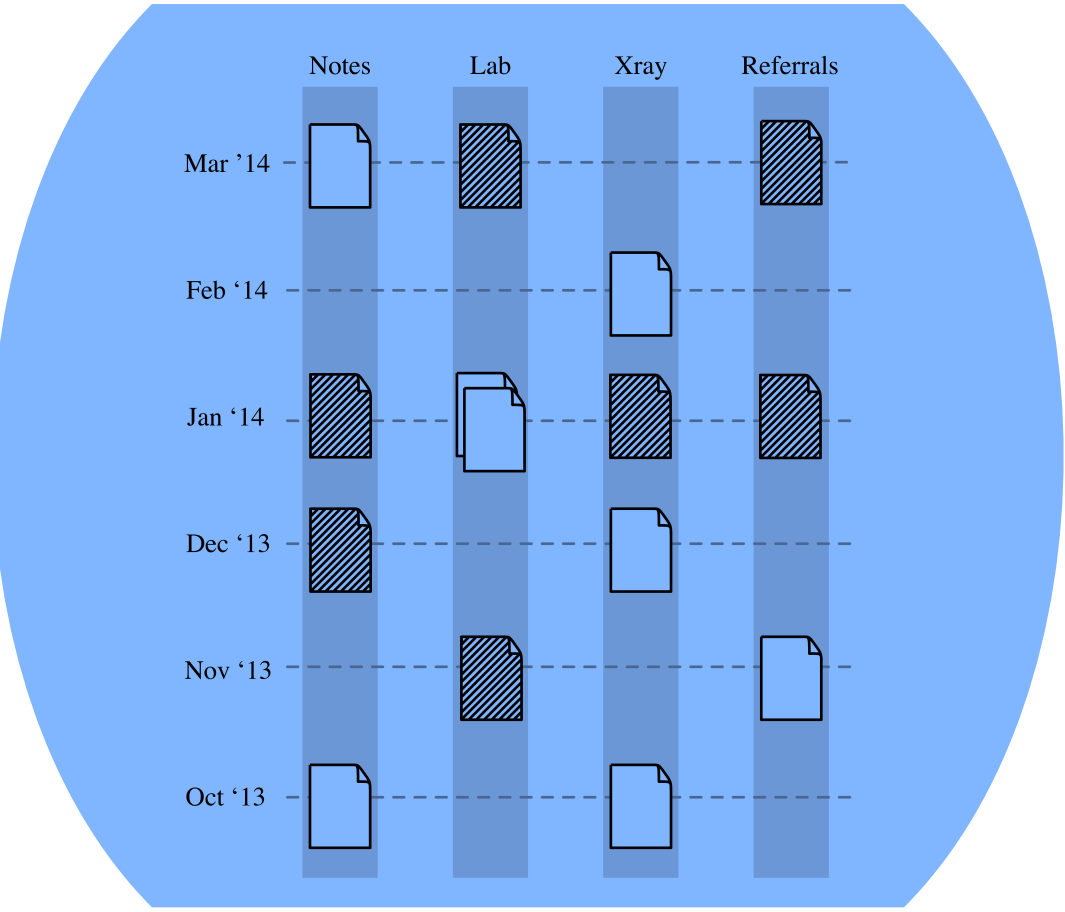


Figure 12.3: Just that one disease is here, here, and here...

The only way to locate the relevant notes is by reading through the stack of notes, top to bottom. Then the only way to locate the other documents that may or may not be mentioned in those notes is by searching through those other stacks from top to bottom. And once you've done that, and

have them all neatly arranged inside your head, you just throw it all away again, since there is no way to persist the relationships you just found. The next time you go through the same process, there's no guarantee you'll come up with the exact same selection of documents from the different stacks, since it all comes down to how attentive you are and how selective, so there is no guaranteed consistency in your view on the record. It's largely a matter of chance. The more documents there are in the record, the less reliably will you be able to find relevant documents.

Having the documents ordered according to the type of document made sense before we had IT system and it was all on paper, since that document could only be in one place[2], and if you had to choose between filing that x-ray among other x-rays, or in a folder "hypertension", it made slightly more sense to file it under "x-rays". Not *much* more sense, just a little bit.

But now, when the number of copies of a document means nothing, it makes no sense at all. An x-ray of the thorax taken during workup for hypertension has its primary meaning as an exemplar of a study of the *hypertension*, not as yet another x-ray. In other words, the fact that it is an x-ray is just an indication of a method, not of meaning. What we actually were looking for was a possible enlargement of the heart, and to determine if that was present, we used an x-ray. We could just as well have done a cardiac ultrasound, or an ECG, but we did an x-ray.

Now, what happens in the head of the doctor when he reflects on the records and wonders if the patient has a cardiac enlargement? What we would prefer to do is go look in some spot labeled "cardiac enlargement", and find the answer: "no, as shown on thorax x-ray", or maybe "no, as shown on ECG", or even "no, as shown on cardiac ultrasound". But with the stack-oriented EHR systems of today, instead I must assume[3] that the possibility of cardiac enlargement has been verified or excluded using *either* thoracic x-ray, cardiac ultrasound, or ECG, and then go searching through each of those three stacks to look for any trace of such an exam[4]. Even worse, if I didn't know of BNP, a new lab test for cardiac failure, I would not go looking in the stack for lab results after this, and would miss it.

What I *really* want to know is if cardiac enlargement or failure has been verified or excluded, and I really don't care how. But what I'm *forced* to do with these systems, is try to read the mind of other doctors, then backtrack from there.

---

[2] Unless we copied it, but that was too much work and too expensive.

[3] That assumption builds on knowledge entirely outside the records, such as conventions in my area, or that the doctor who did the workup came from the same university as me, or that we both ought to be equally up to date, or out of date, so can be expected to use similar methods, etc. There's a lot left to chance, as you can see.

[4] Unless we cave under the pressure and ask the patient, but that defeats the purpose of the medical record, doesn't it?

Most doctors simply repeat findings in the notes to clarify how they drew their conclusions, but this is a workaround for an incredibly poor system design, and it shouldn't be necessary.

12.1 Improvements

There are ways to improve these systems, even though they are fundamentally flawed. These improvements can only be seen as a patch to tide us over until we have more useful systems.

Term clouds

It's very hard to locate information relevant to a particular problem in current systems. But even worse is that you have no high-level listing of which problems a patient has or has had. For instance, a patient could have been investigated for suspected leukemia, and it would be important for me to know that, but since I don't know what I'm looking for, I won't find it. There's no master list of potential problems for this patient in the EHR. The only way I could be informed of problems unknown to me is by scanning through the different document stacks in the EHR, more or less at random.

A solution for this particular problem could be a "term cloud" constructed by extracting all words in the notes in the EHR, removing trivial ("stop") words and then ranking the rest according to frequency. There are refinements on this method, which in principle rate words higher if they occur in places where they would not normally be expected, which improve on the usefulness of the algorithm.

The result can be presented as a ranked list or as a cloud of terms, where the most highly ranked are shown in larger fonts, at the top of the list, or in different colors. Regardless of the exact mode of presentation, this system would make it much easier to see what is significant or unusual about a particular patient record. If the cloud could be projected along a time-axis, one could also get a better view of the development of the patient disease complex over time, which could be particularly useful for multi-pathology in the elderly.

Tagging

A simple method of bringing some structure to the mess that is the classic EHR would be to allow tagging of all elements in the record, such that a tag for "diabetes" could be applied to all encounters where this problem appeared, and applied to all prescriptions, referrals, x-rays, lab reports, etc, relevant to that problem area.

These tags would form the horizontal links between documents in different stacks that are missing, as illustrated in figure 12.3. The advantage would be that one single document could have more than one tag, and then be part of several relationships. A major problem with the solution is that practically all tagging work would be left to the user, who would in general have neither the time nor the inclination to spend effort on this, particularly since any work done would only benefit other users in the future. It's also far from trivial to decide how and what to tag, and we could expect severe differences in tagging behavior between users[5].

Tagging is just a poorly performing kludge, however, since it will easily happen that multiple tags will be created for the same problem, or the same tag for multiple problems. As long as it's used with restraint, however, it may bring some structure to a fundamentally failed data model. Or maybe not[6].

[5] I'm envisioning this resulting in "memoranda" and "standard operating procedures", ultimately ending in "meetings", and maybe even fist fights breaking out between proponents of different tagging theories. What a tragedy.

[6] Kim Nevelsteen remarks: *Indeed there is not a single system that I know of that uses tags efficiently. Requires disciplined users and tag moderation. And, even then some nifty visualization to get it right. Also, it is not certain that humans can form a mental map of a tag cloud like they can for a tree structure or some other organizational system.* I believe that is an accurate assessment.

Part III

Obstacles and bad ideas

Summary

There are a number of obstacles to a better electronic healthcare record, and many of those can be classified as "bad ideas". When the people in power get bad ideas, they effectively hinder any attempts to develop really useful new systems, so we need to identify and describe these bad ideas.

Other obstacles are a lack of clean requirements gathering, false requirements, and a misguided drive towards standards for their own sake. The Taylorization of healthcare, that is the view that it is a production process that can be managed by economic incentives, must be described, and we must find ways to go around that, or even better, eliminate this toxic idea. I'll describe examples of process thinking that is misplaced in our setting.

I'll also briefly go into architectural bad ideas, often promoted by lay people, and forced on the industry. These bad ideas also form a formidable obstacle to efficient development of useful clinical tools.

I can't avoid attacking the user group concept, which is at the source of many bad ideas and fruitless efforts leading nowhere. And lastly, the political poisoning of the purchasing process that, at least in Sweden, has led to enormous waste of effort and money, with very little tangible results.

I'm sorry to say that I end this part of the book with a brief comment on OpenEHR which I sadly also must classify as a bad idea. It may still turn into a good idea, but I think that's pretty far off.

Chapter 13

Research the EHR (fallacy)

Very often we encounter the proposition that a huge collection of EHR records is something we can use for research, with the expectation that we would find new diseases, effective diagnostic tools, and not least, the most effective treatments. This is a fallacy in almost all cases.

Let's first mention what *can* usefully be done research-wise with EHR data, which isn't much. We can use it to signal new epidemics or as an early warning of side effects of pharmacological products, but only as warning flags. If a certain diagnosis appears more frequently than baseline, or in certain geographical areas, we could use this to alert users. The actual determination of what the epidemic consists of, or if the side effect is real, has to be done using other methods.

As far as comparing treatments, the EHR is worse than useless, it's downright misleading, and this use must be avoided at all costs.

To explain why we can find epidemiological triggers but not do research into cause-effect relationships, you have to know the difference between "evidence based" medicine and "anecdotal" medicine.

13.1 Anecdotal medicine

This is how "old, experienced" doctors learned medicine. If they'd tried medication M for disease A in their patients, and it worked 9 times out of 10, they would draw the conclusion that it did indeed work, so they'd keep on prescribing the same medication in the future. The problem with this is that:

- Even 9 times out of 10 may be a statistical fluke.
- The results that the doctor remembers may be the remarkably positive results, while results that interests the doctor less tend to be

forgotten. By using an automated system, this is the only characteristic of classic anecdotal systems we could avoid falling prey to.

- These particular patients may be preconditioned to react well to medication M, particularly as they will tend to belong to a very limited gene pool, live in the same area, and be exposed to the same environment.

- The patients may be influenced by the charisma and conviction of the doctor to feel better even if the medication doesn't in fact help.

- There is no comparison to what would have happened to the same, or similar patient, that was *not* treated, making it impossible to verify that any change is due to the treatment.

- The doctor may be preconditioned to see an improvement in the patient's condition even if there is none.

- The doctor may have counted wrong, maybe it was only six times, but his memory isn't what it used to be.

- Negative effects may be suppressed or ascribed to other factors, by both patient and doctor.

The grand total of all these effects are that our knowledge about cause and effect in medicine doesn't advance much as long as it's based on anecdotal evidence, unless the anecdotal evidence is extremely strong (e.g. strychnine and large bullet wounds seem to kill you).

13.2 Evidence based medicine

Evidence based medicine is mostly based on "randomized, controlled trials" (RCT) which seek to set up an environment so that only the real influence of the treatment is measured. Since we can't eliminate influences from the environment, the gene pool, or the influence of wishful thinking, we can only take care that these influences are more or less equal in a group of patients receiving the treatment we are investigating and another group not receiving the treatment[1].

To make sure we create two groups that differ in no known respect to each other, except that one group receives treatment and the other does not, we have to do the following when setting up an RCT:

[1] The electronics people among you may recognize this as "common mode rejection". Same thing, different application.

- Patients must, one by one, be randomly selected to receive treatment or to not receive treatment (randomized, with a treatment group and a control group).

- Both groups must be similar for all known characteristics such as: social group, gender, age, other known diseases, race, environment (the groups should be matched for these characteristics).

- The patients should not know if they receive the treatment or not, so another treatment, or placebo, must be administered to the control group patients (the study is "blinded").

- The healthcare worker that examines the patient and makes the judgement about results, should also not know if the patient got the real treatment or not (the study is "double blind").

- The selection of the patients must be done before any treatment is done and the study should have specified limits (endpoints) defined beforehand, which define how the results should be interpreted (prospective).

Not every treatment can be double blinded[2]. Operations of different kinds come to mind, where at least the surgeon must know what he or she is doing. Very few operations can be single blind; it's ethically challenging to subject a patient to a placebo surgical procedure that does nothing. It has been done, but it's rare. However, in the great majority of cases, you can do a prospective, randomized, controlled, and double blind study, and that is what you should always strive to do. We know that even if done entirely correctly, we still sometimes draw the wrong conclusion[3]. According to some, it's even worse[4].

On the other hand, if we don't do it this way, we could just as well toss a coin or use astrology.

---

[2] Which reminds me of a hilarious moment when I first read an abstract proposal from a colleague describing a "double-blind study of nifedipine in anesthetized dogs". Think about it.

[3] An often overlooked fact is that a perfectly executed study that shows a difference as a result of an intervention and the calculated "p" is given as "$p < 0.05$", actually means that the chance of the finding being just coincidence, a fluke, is 1/20. In other words, in a single issue of a medical journal with 30 articles, all of which claim "$p < 0.05$", one or two should be expected to show an erroneous conclusion. And this is under the assumption that all the studies were perfectly executed. Which, of course, they aren't, so the situation is actually much worse than this.

[4] http://www.economist.com/news/briefing/21588057-scientists-think-science-self-correcting-alarming-degree-it-not-trouble

13.3 Electronic astrology

Using Electronic Health Care records to find out if a treatment works is another beast entirely. It's not prospective, it's retrospective. It's not controlled. It's not double blind, not even single blind. It's nothing more than automated anecdotal medicine. As the saying goes: *with computers, you can make the same mistakes much faster and on a larger scale*[5]. Don't.

13.4 Exceptions

There are a few exceptions to the rule, situations where data can be used from an electronic healthcare record system to signal new discoveries.

National registries

By registering measures of outcomes from different hospitals or regions, including differing methodologies, large differences can be detected and studied further. The key here is that the indications obtained this way can generally not be used to draw conclusions about methods, but only to signal that a properly conducted scientific study could be warranted. And even then, we risk directing resources to a problem that happens to be more visible using these methods, without therefore being more important in a more objective scientific sense.

Quality control

Followup of the performance and longevity of implants of different kinds could be meaningful, and data for this could possibly be extracted directly from the EHR system. Examples of successes involve determining which kind of hip replacement implant and implant method lasts the longest or has least complications.

Epidemiological triggers

Systems could signal unexpected correlations of incidences across records or systems, such as a suspicious number of very similar infections within a hospital or region. Any such signal would have to lead to a scientific investigation in its own right, but the signaling as such by the system could be very useful.

Neither of these exceptions lead directly to new science. All three need to be complemented by properly conducted studies.

---

[5] "*A computer lets you make more mistakes faster than any other human invention in history... with the possible exception of handguns and tequila.*" –Mitch Ratcliffe

In other words, the role of the EHR as a tool for scientific discovery is very limited and should almost certainly remain so. Resist the urge to buy or sell these systems as research tools. Please.

Chapter 14

How management data corrupts

Remuneration schemes in general are either utility-based or cost-based. In other words, prices are determined based on the cost of production plus a margin, or prices are determined on how much a consumer is willing to pay. For example, if you are employed, you are being paid on the basis of your use to the employer, not on how high your costs are for living, travel, vacation, and iPads[1]. At the other extreme we have taxes, which are based on how much it costs the state and county to provide services to the population on average.

When it comes to remuneration in medicine, we see all kinds of models. Some payments to healthcare providers (doctors, GP centers, hospitals) are based on cost, a budget that is put together to cover expected material costs and salaries needed to provide a certain level of healthcare for a certain period. Other systems are based on results, i.e. utility, under the presumption that rewarding quantity and quality with money will improve results.

In both cases, cost-based or utility-based remuneration, the EHR will fill the role of data provider for the calculation of the remuneration. It's obvious that if we aren't careful, the data put into the EHR will be distorted by attempts to maximize budget or payments to the provider, greatly diminishing the veracity (integrity) of the medical data in the EHR.

If the size of the budget, or the size of payments, is determined by the "case load", which is a measure of how much care the patients need, the coding for severity of diseases and complications will tend to be exaggerated in the EHR. If the size of payments instead is keyed to the results, complications tend to be suppressed and results glorified. If costs are emphasized instead, the costs will either be reduced beyond the reasonable, or great effort will be spent moving the cost to other providers or departments.

[1] The application of minimum wage standards are the exception.

In all these cases, the data in the EHR will be skewed and manipulated to reflect a more "budget-friendly" version of reality, making the EHR data less reliable for its original purpose, namely to provide better healthcare. This implies that we ought to ban any budget related measurements of healthcare from using the EHR for data collection. Yes, this must be a horrible realization for management, but there is no other way to safeguard its usefulness for actual patient care.

> *When designing an EHR system, it's imperative to avoid building in incentives to manipulate and corrupt the data to optimize for economic results. The power of the budget is underestimated at your peril. Even stronger, it ought to be prohibited to use clinical data for economic reasons, since there is no way to do that without undue and destructive influence on the contents of the record.*

As a side note, it's amazing how little influence these management tricks have on actual care. The influence seems limited to the corruption of the documentation. The question is if this is sustainable, or if future doctors and nurses will choose to treat patients differently depending on rewards, leading to disasters. Our political overlords seem not to take this risk into account while playing their games.

Chapter 15

Beware of false requirements

When defining healthcare IT systems, be careful to formulate correct requirements and avoid "false" requirements, which are solutions in disguise.

Before building a system, the requirement engineers elicit "requirements" from the users in some fashion. There are three major ways of doing this:

- The requirement engineers *are* the users. This can only work if the users are experienced in both the target domain (in our case, medicine) *and* IT requirements gathering and design. Such users are rare.
- The requirements engineer observes users in their natural habitat long enough to understand how they work and what the essential requirements are. This rarely happens in medicine, but I don't know why. It could have to do with fears related to patient privacy, but that is not as difficult to overcome as many seem to assume.
- The requirements are gathered from a user group. This is the easiest, and by far worst, solution. Practically all requirements gathering in healthcare projects seem to take this route. I'm will discuss the perils of user groups in more detail on page 133.

Most poorly expressed requirements stem from using the wrong source for the requirements, in this case user-groups, to define them. But even if you used the right source, you have to be careful how the requirements are formulated, so that they convey exactly what is needed and nothing more than that.

A real requirement defines something the system should do. Not *how* it should do it, but only *what* it should do. The requirement expresses

a goal, a desirable outcome, or a defined limit on that goal. The less the requirement says about the way to achieve this goal, the clearer the requirement becomes and the more technical freedom is provided to the system architects to find an answer. The need expressed as a requirement can be viewed as question, and the resulting design as the answer. A well formulated requirement should not give any hint as to what that answer should be, but it should be very clear when an answer satisfies the question. If you see requirements with hints of solutions in them, you should do as TV lawyers do, stand up and shout: "Objection! Leading requirement!".

This is a larger problem in healthcare system design than in most other areas, because the system architects and designers are less able to detect false requirements. Most system architects and designers know something about riding the bus, accounting, paying with credit cards, or even management of traffic lights, so will automatically and without second thought see what part of a requirement in any of these areas is unnecessary or misleading, and just ignore that. But when it comes to medicine, the distance between the domain expert and the system architect is much larger. A lay person has a really hard time knowing what is real and what is misleading fluff when it comes to medical records requirements. If we want to avoid designing for the fluff, we have to avoid having that fluff in the specifications in the first place.

For instance, if a doctor defines a need that he should see all the notes by all specialists in a list to the right on the screen, what can you as a system architect or designer conclude from this? Most architects and designers will quickly decide that "a list to the right" is a false requirement, based only on what that particular doctor is used to seeing. It is also fairly obvious that the doctor is thinking about a desktop screen, not a tablet, unnecessarily limiting the solution space, so the part about the "right side of the screen" will probably be ignored as well.

But the real kicker is that most architects will *not* question the idea that the doctor needs to see all the notes, which is the major false requirement expressed here. What the doctor *actually* needs is to be made aware of all the major diseases and problems the patient has, and due to the shi... um, suboptimal systems he has always worked with, so sees no other way of achieving this than to be presented with *all* the information so he can wade through it himself, trying to make sense of it. This false requirement is the expression of what *the doctor thinks can be done*, since that is all he has ever seen until now.

If the architect takes this requirement at face value, the system he creates will be just as fundamentally flawed as the doctor who formulated the requirement expected it to be.

If done correctly, however, the requirement could have been expressed as: "the user should be made aware of all the major diseases and problems the

patient has or has had". With this rephrasing, we pared the requirement down to the essential need, which allows an architect to consider different solutions to the problem, unencumbered by any erroneous presumptions by users. Yes, presenting a list of all the notes is *one* design solution satisfying this requirement, but it's not the *only* solution, or even by far the best. With this improved version of the requirement, the architect and the designer will have the space to think up a few alternative solutions, then present them to the user for verification, resulting in a much better design.

The problem with false requirements, in short, is that they needlessly limit the solution space. Let's discuss a few more examples.

15.1 One patient - one medical record

The requirement "one patient - one record" is often postulated under the dual assumption that "things" are better if all information about a patient is available all the time and to every involved doctor, and that this implies a common system. There are three unproven and undefined parts in this false requirement:

1. The "things" that will improve are desirable and appropriate, even though these "things" are never defined. See my discussion elsewhere in this document about "Where are we going with healthcare IT".

2. That the care of the patient improves if all information about the patient is available everywhere. And, implicitly, that the improvement of the care is highly significant[1]. So significant, in fact, that it warrants rebuilding entire systems.

3. That to make all the information available, a common system is necessary and sufficient. If that was the case, AOL would rule the internet (see page 201 for more on this example).

Actually, none of these postulates are true.

> *But what's worse is the unintended consequences of this false requirement, namely the ruling out of any solutions that do not involve humongous stovepipe systems, stifling innovation while leading an entire industry up the garden path.*

[1] See in particular my discussion about encapsulation and separation of concerns on page 181. There is every reason to think that the availability of the entire corpus of detailed information about the patient at every point of contact actually makes healthcare much worse, not better.

Closely associated with the false requirement of "one patient - one medical record" is the false requirement of "one common system". The underlying *real* requirement is effortless interchange of data between different healthcare providers, the need to make doctors aware of significant healthcare related issues regardless of which provider created information about them. The assumption that leads to the requirement for a single system, is that interchange will be easier, even trivial, if everyone uses the same system from the same vendor. We know from other domains that this is simply not so.

Large vendors have everything to gain from reinforcing this fiction, since that would enable them to increase the total share of sold software to the provider, while not having to partake in any discussions with other vendors for integration with other systems. Purchasers also have much to gain from this idea, since it moves the responsibility of making parts work together away from them to the vendor.

Small vendors, on the other hand, usually argue that it's better to build up the whole from independent parts from several vendors, else they'd have no market.

Both of these arguments should be taken with a large grain of salt, since they're both self-serving. A political organization (chapter 21) will always tend to listen to the large vendor, since the vendor, in exchange for getting a large contract, promises to take on the responsibility for the whole, such that the purchasing organization needs not step up and take the blame if something goes wrong. This doesn't in fact diminish the risk of things going wrong, but diminishes the risk of the political organization to find itself in the cross-hairs of criticism, which seems to be of higher concern than the actual functionality of the system.

The real requirement behind all this misery could be formulated as: "avoid risk of work and blame for the management organization". If expressed in that way, other solutions could be thought out and presented. Such as improving the management organization, for instance.

15.2 A common database

We see this "requirement" quite often as a requirement to integrate with a predefined database. This database is sometimes even a separate project set up by the customer, see for example the GVD[2] project in Stockholm. The whole thing can be viewed as a downsized version of "one patient - one record", giving the illusion of a free market for competing applications. The idea was that vendors could all freely compete, each with their own

[2] "Gemensam Vårddatabas", eng: "Shared Healthcare Database", a government led project to unify the underlying database used by independent application providers.

applications and features, as long as they used the GVD to persist and to read data. Since the major difference between these systems today are in the data structures much more than in the user interface or processes, this idea would make most applications look almost identical. Clearly, the vendors were not interested; not a single one signed up for the project (at least as far as I know).

Interestingly, when reading the critique of the GVD project, as in the HL7 Watch blog[3], one is struck by how management blames exactly the wrong reasons for the failure. They seem to think it was the wrong type of database, while the real problem is that a common database is the wrong idea[4].

A common database cannot reasonably be a real requirement, however. It's an architecture, or a design solution, but not a requirement. Since there is no way an end user can detect if the information is kept in a common database or not, it *can't* be a requirement[5].

On top of all that, it is also one of the worst ideas for architectural solutions I've ever seen. I can't, off the bat, think of more than maybe one example in the universe of vertical[6] IT solutions where different applications meaningfully and successfully work with a common database, without those applications really being just plug-ins or scripts inside a larger application.

The only example of a pure single database with different competing clients I *can* think of are Twitter clients, where there is a range of different user experiences, all based on the same database and service objects. But the essential Twitterness of all these clients is clear. There's not much to make one Twitter client essentially more functional than any other, simply because they're all based on the same database back end. So, yes, for a very limited application, a shared database could work.

Service oriented architectures come the closest, but even then, there is a distinct service between the database and the applications that use the data. Entirely different applications tend to use different service layers as well. In those cases where a number of different applications are designed

[3] http://hl7-watch.blogspot.se/2008/03/news-from-stockholm.html

[4] Actually, I don't think that's how they really reason. I'm pretty sure they're just assigning blame away from themselves. The problem with that is that the mistakes they made will be repeated as long as the real reason for the failure remains obscure.

[5] Someone somewhere made the point that a criterion can only be a real requirement if its implementation makes a difference to the end user. If the end user cannot know if the criterion is fulfilled or not, it is not a requirement. Too bad I forgot who said this. By the way, the "user notices" is a necessary, but not sufficient, characteristic for something to be a requirement.

[6] "Vertical" solutions are solutions covering a particular domain with a number of different kinds of tools, just like EHR systems that contain tables, text documents, communication tools, etc. "Horizontal" solutions are particular tools, such as a word processor, that can be used in a large number of domains, such as healthcare, accounting, book writing, etc.

to use *the same* service interfaces, these applications tend to largely consist of other functionality and then use these interfaces only for a relatively limited set of features.

15.3 Terminology systems

We often see a formulated requirement that medical systems must work with a standard terminology catalogue such as SNOMED CT, but again, this is a solution and not a requirement. If we try to reel back the thread that connects this solution to the requirement that it ought to be hanging from, we get nothing. It's a solution looking for a problem which hasn't been found yet[7].

A major reason behind creating a fully codified medical record is the presumption that it would enable the computer to "understand" the contents and then contribute to better healthcare somehow. I discussed this briefly in the introduction on page 4, and I won't repeat that reasoning here.

Another major reason to introduce coding is to make the record contents better defined, less ambiguous, and more amenable to research. When reading records as they look today, we'll find a lot of waffling going on. Doctors often seem to avoid making clear and unambiguous statements about what they see and what they decide. The reason for this is that doctors aren't all that sure of everything, and regular human languages are pretty good at giving us a range of methods of expressing that uncertainty. To researchers, that uncertainty is a real difficulty, making it hard to use as input data. The desire to make the record much more defined is understandable, but forcing users, by means of coding systems or user interfaces, to be more exact in documentation than they are in reality only leads to frustration, missing data, or even worse, wrong data.

> *Using coding systems to force clarity in an unclear situation will only make the data look good, but it won't make the data right.*

[7] Since I originally wrote this, I've started to use SNOMED CT in iotaMed to good effect, partially solving the problem I had with translations. But I still haven't seen any *other* real problems with a similar solution yet.

Chapter 16

Avoid the need for standards

Note well, I'm *not* saying you should avoid standards, I'm saying you should avoid the *need* for standards.

16.1 Lab codes

If you want two systems to communicate some lab value, they probably need to agree on how that lab value is identified and how it is measured. Or maybe they don't. If the value is to be presented to a doctor, and that is all it is going to be used for, there really is no need for the system to have the faintest idea what is going on, as all it needs to do is present the name and the value as a string and leave the interpretation of the value to the user.

If, on the other hand, the system is supposed to order all incoming results of a specific kind and display the values as a graph, then it needs to "understand" which lab test is which, and which ones belong together. However, this situation is much rarer than you'd think, so in most cases there really is no need to match different lab tests against each other.

But, for the sake of argument, let's pretend there is. Let's assume that you need to be able to identify the same kind of lab test from two different systems as the same kind of test, and further, that the tests are actually similar enough to be regarded as of equal value. How will you then design a coding system that will match these two?

One way to go is to name someone responsible for maintaining a common coding scheme, and then hope that everyone else adopts that coding scheme for their own use. Any similar tests would then, hopefully, have the same code in all systems[1]. Even though this has been tried many times over the

[1] A lot more is required before two measurements of the same kind from two different labs are directly comparable, but let's forget that for now. We're just pretending, anyway.

years, leaving a trail of abandoned standards along the way, like so much roadkill, labs still use a whole zoo of different coding tables.

Now, one can argue that interconnecting systems would be easier if all labs used the same codes, but the extra effort needed to achieve a universal code and keep all the systems up to date would probably overshadow any savings in integration work. Think about it this way: if a lab changes a code in its own tables, it needs to update the interface to all its customers using that code. But if a standard changes a code, all systems need an update, regardless of if they need that code or not[2].

16.2 Terminologies

Healthcare record systems utilize a great number of terminology systems. Those that are useful are usually limited in size, often containing a few thousand items, of which the average doctor needs to know tens or hundreds. Examples are ICD–10 for disease and therapy classifications and ATC for grouping of pharmaceutical products. Such terminology systems are great and have shown their usefulness over and over.

But then we arrive at current efforts to standardize the actual terms used in medical records for findings, examinations, and conclusions. The main contender is SNOMED CT with around 400,000 terms. Yes, you read that right. 400 ton of terms. The sheer size of this list should tell you something isn't right here. This idea is fraught with problems, not the least of which is that there is no obvious use for such terms. Let's check up on the reasoning behind these common terms.

Making the computer understand the EHR

If I write this in an EHR: "Patient is coughing, has a fever, and I wouldn't be surprised if he's got pneumonia just like his father had last week", and I write in another record: "The pulmonary infection is slowly improving", I am talking about the same problem in different terms. It may be hard for the software to determine with absolute certainty that I'm talking about the same problem, and even worse, it is hard for the system to parse out if the patient *has* the problem, if he is getting worse or better, and finally, what made me conclude that he has pneumonia and what I decided to do about it. It would be much easier for the system to understand all these relationships if I expressed myself in a stricter fashion, using one of several established syntaxes to describe what is medically going on.

[2] Yes, I'm overdoing this argument and it's not entirely accurate, but it's snappier than the full explanation.

The problem here appears to be how to make the system understand what's going on, but the problem in actual fact is *why* we would want the system to have such understanding. We'll get back to that.

Is the data reliable and complete?

No, it isn't. The reason for this is found in how records are written. During a patient encounter, this is what usually happens:

1. The doctor reads up on the patient from the records. If available, he scans summaries of the patient's history. In a fearsome number of current systems, no such summary is available, so he reads notes for as long as he can bear it and his attention span reaches, which is usually 30 seconds to a minute or two.
2. The doctor, if he is smart, asks the patient for major history points he may have missed while reading the records.
3. The doctor asks the patient about his current problems, if any.
4. The doctor examines the patient, jotting down notes on a paper or doing his darndest to memorize key findings.
5. The doctor draws his conclusions, discusses them with the patient, decides on a plan together with the patient, writes documents and prescriptions and sends the patient away.
6. After the patient is gone, the doctor dictates (or writes himself) the notes, including findings and conclusions.
7. Goes out to get the next patient or goes to lunch.

As you can see, the doctor dictates or writes down the notes in the healthcare records only after drawing conclusions and discussing them with the patient. In other words, he reaches his conclusions before writing down the data that ostensibly led to those conclusions, reversing the order of cause and effect.

> *The result will practically always be that whatever findings he writes down or dictates will be relevant to, and supportive of, his conclusions. You will almost never find unexplained or contradictory findings in records written this way. The findings will always support the conclusions.*

This leads to an extremely important corollary:

A correctly programmed computer will never be able to draw any valid conclusion from the health care record that was not already drawn by the doctor, since there is no data available in the record supporting any such missed conclusion.

16.3 What is the benefit of common terms?

But now we run into a problem: *why* should the system need to understand what is happening to the patient, medically speaking? The reasons as claimed by proponents of coding are, as far as I am aware:

1. to increase ease of communication between systems
2. to allow the system to automatically make diagnoses missed by the doctor
3. to force the doctors to express themselves clearer and with less ambiguity

Let's murder those arguments one by one.

Increase communication between systems

There is no obstacle to this communication today. Journal text is transmitted as plain text which is just as useful for a doctor or a nurse as text where every keyword is recognized by the computing system. As long as the formatting and alignment isn't just plain awful, it doesn't matter one bit if the program recognizes a term or not[3]. Actually, it won't even be possible for the user to determine if the system has "recognized" a term or not.

Recognition, or "understanding", of the content by a computer program consists of two parts:

- The recognition of the term, and optionally the expression the term is part of.
- The execution of a action of some kind, triggered by that recognition, which results in a change in state of the program or the data.

If the recognition is missing, nothing will happen. But if the execution is missing, nothing will happen either. There is simply no point in providing recognition without an action of some kind, and so far, in this drive towards

[3] If you paid attention, you recognize this as the characteristic of a design element that is *not* a requirement. QED. For more see page 99.

standard terminologies in medical records, I've never seen mention of any effort to define these actions.

All of which leads me to conclude that the recognition of a term by the system leads to no detectable change in behavior of the system, so it is not needed.

To allow the system to find missed diagnoses

If you look back to the previous section (page

> *The only way I can see we can provide the system with sufficient and impartial data for automated decision making is to feed it using the medical tricorder*[4]*, which, sadly, hasn't been invented yet. Or, subject every patient to an hours long all-inclusive clinical examination by a "doctor" who can do every clinical examination in the book without understanding the meaning of any of them.*

Forcing doctors to express themselves with less ambiguity

The lack of precision in medical records is often a source of frustration to non-doctors. They can't understand the waffling that is always going on, all the different ways something is said, or is almost said. On reading a medical record, you'd be pardoned for regularly crying out: "*Does* he or does he *not* have an abdominal problem requiring a laparotomy...?!" or something to that effect. This is then presumed to be easily solved by forcing doctors to come out and make a clear and definite choice when recording findings and conclusions and not let them get away with vague language.

If the cause of the vagueness was the vague terminology, there would be some value in this reasoning, but we're not that fortunate. The cause of the vagueness is fuzzy thinking, not fuzzy note-taking. When you can't figure out what a doctor was really thinking from the records, it's because *the doctor himself* couldn't figure out what he was thinking when he wrote it.

> *Forcing the doctor to choose between two clearly different and mutually exclusive findings or conclusions against his will, will only result in him either not writing down anything at all, or him choosing one of the alternatives more or less at random. In either case, the utility of the data will have become drastically*

[4] If you don't know what this is, ask around until you find a Star Trek fan. He will explain it to you.

> *worse, while at the same time looking better. This is a clear case of induced false precision.*

And this, dear reader, is the worst thing you can do to otherwise respectable data. The uncertainty in the findings and the diagnosis is of at least as much importance as the actual diagnosis itself[5]. By forcing a terminology on the doctor, you filter out the element of uncertainty and the attribute of open-endedness[6], thereby probably removing the most important part of the information. The solution could theoretically be to incorporate a measure of uncertainty to complement each term, but the complications of using such a system would quickly get out of hand[7].

The ability of common language and discourse to express findings and conclusions, together with exactly the right measure of ambiguity and uncertainty, cannot be replaced by a synthetic language without becoming completely unwieldy and useless. And even if it could be done, it would replace the current ambiguity with a synthetic ambiguity which would not have resolved the problem it set out to resolve. The problem is that reality isn't what we'd like it to be. Tough luck.

> *The ambiguity in medical records do not stem from a lack of terminology, but a lack of ability of the doctor himself to pin down the findings or the conclusions with more certainty. A terminology change does not change this root cause. It may only hide it.*

Too many synonyms

There is another part in the motivation for common terms that is simply ludicrous and that is that doctors use too many synonyms for the same diagnosis or finding. A common term list would force us to use one and the same term.

Well, seriously, if you had the choice between training tens of thousands of doctors to use the same term or program a computer to match say five different terms to one, which would you think is more rational and expedient?

---

[5] Inspired by a similar deduction in a non-related area in the section "Don't cross a river if it is (on average) four feet deep" in "The Black Swan", N.N. Taleb, 2nd edition. As I read it on Kindle, I can't give you a page reference.

[6] My spell checker insists there is no such word, but I'm not going to let my spell checker run my life.

[7] You can do that in SNOMED. I once found a powerpoint presentation with a total of 28 slides showing how to encode a systolic murmur found during auscultation of a horse heart. 28 slides. The author wasn't ironic, he was proud of the result. Think about it.

Curiously, this effort to merge clinical terms into a smaller set is more difficult than one would expect. Terms turn out to have subtle differences in meaning, making it difficult to just dump them into the same bucket. What you have to understand, though, is that these subtle differences in meaning are absolutely essential and should not be forcibly eliminated. They're part and parcel of how medicine works.

Chapter 17

The oversimplification of processes

Just as the medical terminology is forced into a straightjacket by wishful thinkers, so are medical processes. Just witness the following diagram "borrowed" from SKL[1]: see figure 17.1. The model comes neatly labeled in Swedish, so I need to translate the labels into English for you.

The numbered labels are "process steps", while the alphabetical ticks are "measurement points". Let's go through the "process steps" first:

1. Submit request for care.
2. Assess request for care.
3. Assess need for activities. (Yes, really.)
4. Plan execution of activities.
5. Execute planned activities.
6. Evaluate activities and determine remaining need for care.
7. Terminate care process.

Here's my translation of the "measurement points":

A. Decision to submit request for care.

B. Request for care received.

[1] SKL = "Sveriges Kommuner och Landsting", an umbrella organization for regions and counties. The diagram can be found at http://www.flodesmodellen.se, a site dedicated, it seems, to this diagram and the idea behind it.

C. Making a decision about the request for care.

D. Decision to perform activity.

E. Start of activity.

F. Decision to end care process.

G. Termination of care process.

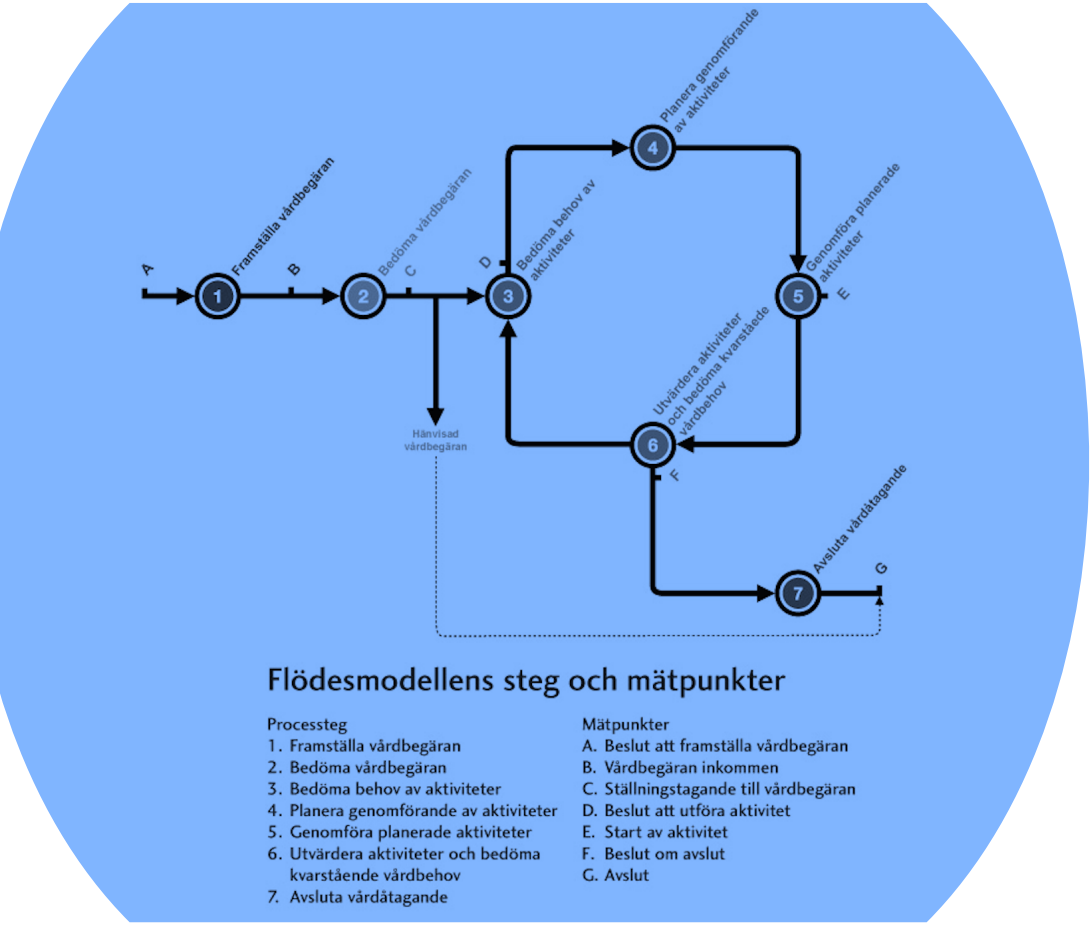


Figure 17.1: The "flow" model from SKL

Even without understanding every label in this diagram, you get the general idea: the patient has a need, the need is qualified and quantified,

and if real you enter a treatment cycle consisting of planning, execution, evaluation, and possibly modification before another iteration, until done. Nice. At least it would be nice if reality deigned to conform to the diagram, but it doesn't.

The whole idea is seems to be based on Taylorism, a management principle used in factories to measure the production process as accurately as possible, then providing the workers with the right incentives to increase production. This management method needs precise measurements, so healthcare management decided to measure number of people coming into the system, the number being in the process, on the hospital floor so to speak, and the number of live people leaving the system, *and* of course, the number of dead bodies dropping out the bottom of the diagram. What could be simpler?

Then maybe, if the ratio of dead people to survivors goes up too much, suitable incentives could be introduced to motivate the healthcare workers to cure more and kill less. But not too much, that would be uneconomical; we're looking for optima here.

This is clearly a system only a civil servant could love. And that's ok, as long as these people don't run the system, but when they do, you have a problem.

17.1 My amended version

The SKL diagram obviously has very little relationship to reality, so I'll help them out with the missing flows. Adding in a few of those, the diagram should look more like in figure 17.2.

Even though my modifications may look like a joke, they are serious. I can give several real and daily examples of every one of those red arrows I added (and I will at the end of this chapter). The thing is this:

> *If a process is inherently non-linear, but you need a linear model to implement a tool, you will fail however hard you try. Instead, embrace the non-linearity, especially since you have no choice. Your linear model would only describe a reality that doesn't exist and your tool will solve a problem that does not exist, in a way that won't work.*

Let's just take one example from the diagram. See that branch going from state 6 to state 7[2]? It says at state 6 to "evaluate and determine remaining

[2]It's not clear from the diagram which parts are states and which are transitions. There's numbers and letters all over the place. If somebody can figure it out, please let me know. Or maybe not. Who cares, really?

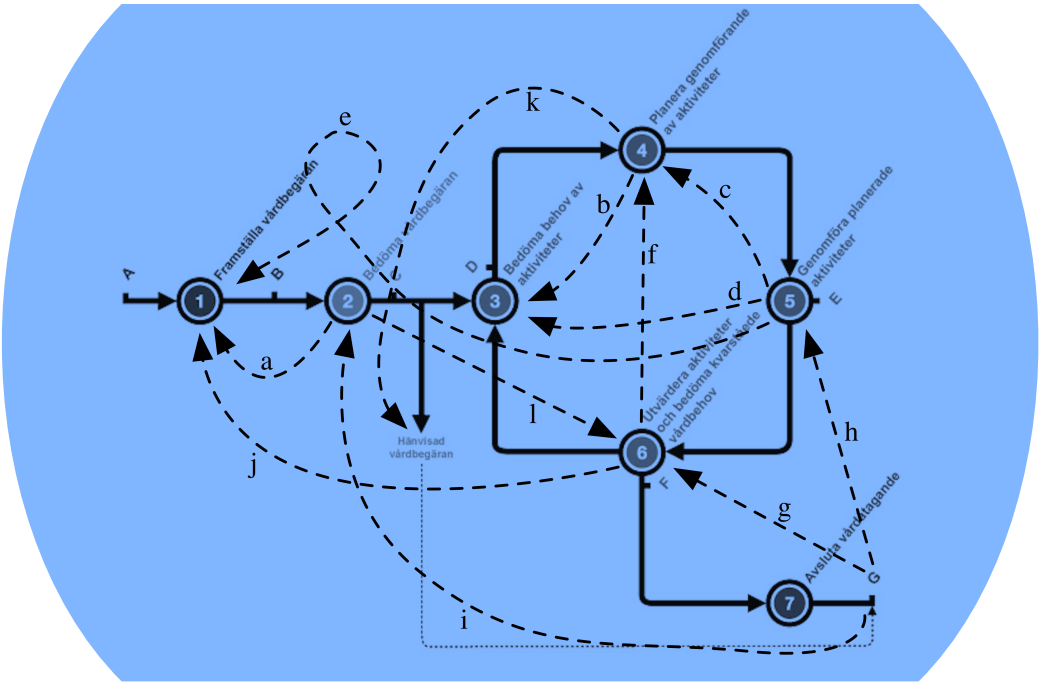


Figure 17.2: The flow model with my amendments

care need" and at state 7 "terminate care". The entire diagram rests on the assumption that care is either given over and over or finally ended, but there are hardly any real world examples of this. When we "end" care, it's practically always a case of "suspended further care until something happens again", which it may never do. The only real end state is the death of the patient[3], since we do not include resurrection as a possibility.

Think of cancer, for example. When is the care ended? After five years of no disease? Ten years? Let's assume we have two patients, one has a recurrence after nine years, the other after 11 years. If we set the limit to ten years, the first guy is still in the iterative part of the flowchart for his first cancer. The other guy is, according to SKL's diagram, either a victim of a new cancer, or is a non-existent patient. What are they going to do? Ship him off to Norway?[4]

[3] And even then... imagine having erroneously "terminated" a patient. How do you recover from that state?

[4] We could ship out our undead patients to Norway, and let Norway ship theirs back to us. If, that is, they have as crazy "flow models" as we do.

This diagram, and the thinking behind it, reflects a call-center mentality that does not fit medical processes very well. Imagine having your cancer diagnosed and treated by Dell's customer service, and you get what I'm talking about.

17.2 Labeling the disaster

I promised to explain what my modifications exemplify. So here are my additions to the diagram:

a. Goes from state "2" to state "1", against the stream, so to speak: while assessing the request for care, we find it incomplete, reject it and send it back for re-submission. Very frequent, and very irritating to have happen to you.

b. Goes from state "4" back to state "3": while assessing the need for activities we discover that another activity needs to precede this activity. Back to assessing the request.

c. From state "5" back to state "4": while performing an activity, the activity fails for some reason and needs to be planned again for later.

d. From state "5" back to state "3": while performing an activity, it is discovered that it is the wrong activity, so it needs to go back to assessment.

e. From state "5" all the way back to state "1": during the execution of the activity, a need for another request for care becomes obvious.

f. From state "6" back to state "4": during assessment of an activity, we can discover that a companion activity was missing from the planning and immediately plan that missing activity.

g. From state "G" (yes, I know, is it a state?) back to state "6": after the end of the care process, the patient comes back still having the same problem, so we need to pick up the activities where we left off.

h. From state "G" back to state "5": often we terminate a care process while giving the patient the necessary prescriptions or referrals to let himself get further care. Examples: writing a prescription for antibiotics, and asking the patient to go only take them if he runs a fever.

i. From state "G" back to state "2": after terminating care, we may go back to reassessment of the request for care if the problem doesn't stay solved.

j. From state "6" back to state "1": while assessing the activity, we may discover there is nothing to treat.

k. From state "4" back to state after "2" (unlabeled!): during planning of activities, we may discover we need to refer the patient elsewhere.

l. From state "2" forward to state "6": if we receive a referral for treatment of a patient that we already have had in treatment for the same thing, we go directly to assessment of the activity, and pick up from there.

Have I made my point yet? Yes, I think I have.

Chapter 18

How to connect

When systems need to share data, there are different ways of doing that. First, we must consider the need for encapsulation (see page 181), but if you promise not to violate that principle, then we can discuss integration aspects.

Let's assume we have three systems which may or may not be of the same type.

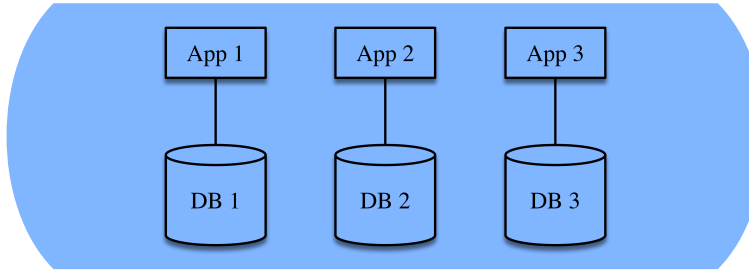


Figure 18.1: Apps need ways of talking to each other.

18.1 Big fat and ugly

These three systems can be integrated in different ways. The first and most obvious way is to simply share the same database for all three systems. See figure 18.2.

As you may figure out from the title to this section, I'm not all that crazy about this idea. The problem is that the three apps will turn out to be three minor variations of the same application. Medical applications don't do much processing or use many distinctive algorithms, so the application's

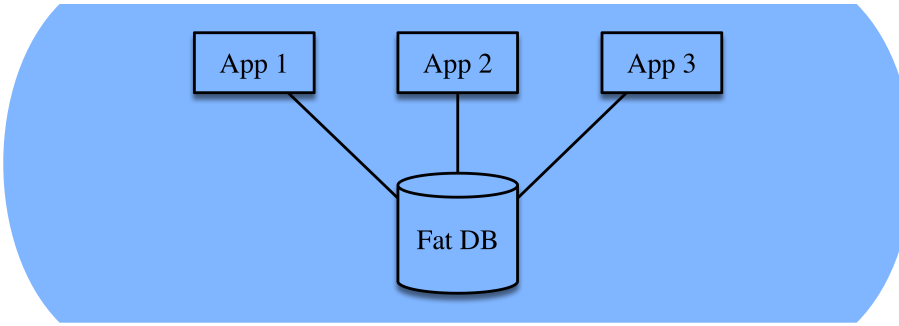


Figure 18.2: Common database as medium.

functionality is almost entirely based in the originality of the data structures, and if you limit the data structures to one common structure, there is no place for competition in the remaining parts of the applications. Hence, no sensible application vendor will ever want to work with the common database[1].

The idea behind this design is to reduce the data interchange problem by making all data equal to start with, so there is nothing to interchange. But this can't work, since some data will always come from outside the system. The major disadvantage to huge systems is their inability to change and adapt. For medical systems, that is one of the most important abilities to preserve, since the number of data structures and their number of constituent attributes rise as medical science advances and new diagnostic and therapeutic methods are invented, so there's a serious cost of inflexibility to unified medical records systems. See the section on "What is the difference?" on page 129 for a more elaborate discussion of this aspect.

18.2 Inter-app messaging

The next idea, inter-application messaging, is what we see used in most systems today (see figure 18.3). It is based on the interchange of messages at the application level (as opposed to the database level). These messages usually pass through some sort of routing engine, where they get sorted according to source and destination, but also can be transformed from one message form to another.

Generally speaking, the engine can only transform messages based on the information present in the incoming message. It's hard for these systems to

[1] IMNSHO, this was the major reason for the very expensive failure of Stockholm's GVD project. See page 102 for more on GVD.

consolidate information from different sources, or from the same source but from earlier or later messages. They are, in general, what we call "stateless" applications.

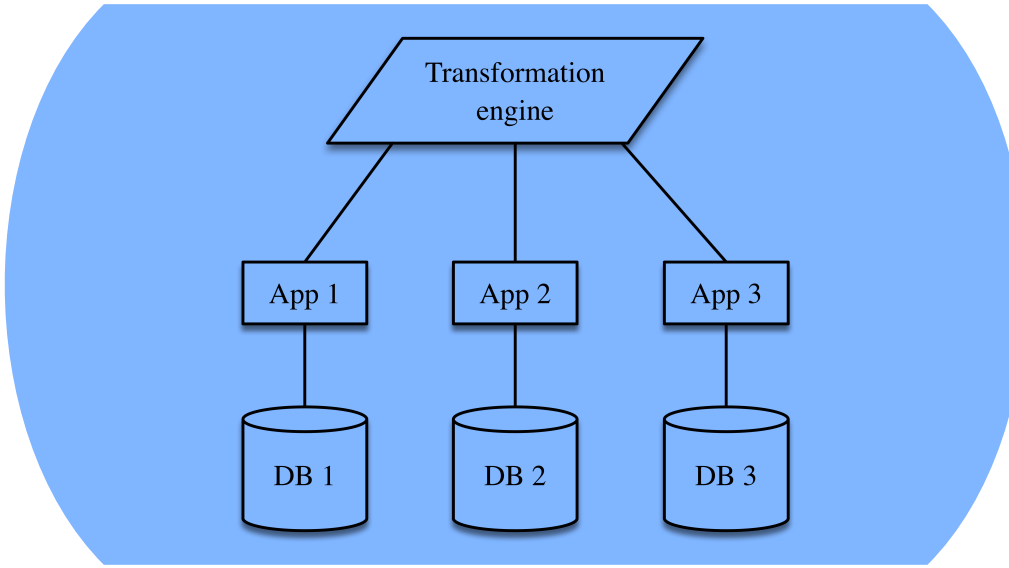


Figure 18.3: Messaging between the applications.

Too much dependency on this model also makes it unnecessarily complicated to communicate within smaller organisations. This model requires quite a bit of standards to be really effective, and too much reliance on standards makes for expensive and slow work[2].

18.3 Inter-database messaging

My third model (figure 18.4) is based on the interchange of data between database instances through intermediaries I call blackboards, since they *are* blackboards in an architectural sense[3] ("BB" in the figure).

[2]When a standard is needed, I'm all for it, but I'm even more in favor of avoiding the need for a standard if possible. See page 105 for more on this.

[3]The "blackboard" pattern describes a design where a number of different systems read and write from a common "slate" or "blackboard". The advantages include decoupling of formats *and* decoupling of activities, making multitasking, load balancing, message transformations, and redundancy much easier to achieve. One nice description can be found in http://st-www.cs.illinois.edu/users/hanmer/PLoP–97/Proceedings/lalanda.pdf, but there are many more.

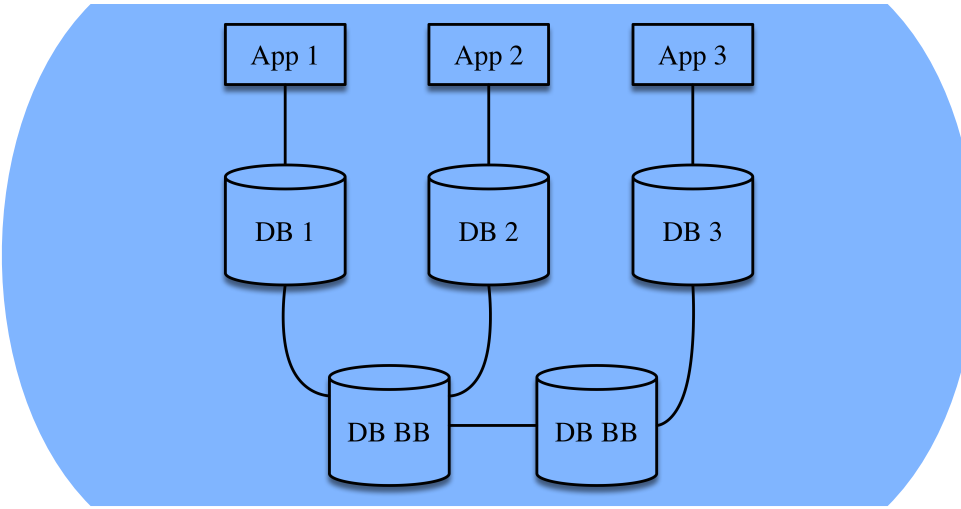


Figure 18.4: Messaging between databases.

While moving the messages, conversion of formats can occur. The converting process is close to both databases, so it can look up missing data from other sources in the database, even if that data was created at another point in time.

This design achieves most of the advantages of the "big fat and ugly DB" above, with a cheap and resilient data design, and without the other drawbacks. In particular, huge unified systems are no longer necessary, or even a particularly good idea. This design replaces practically all the claimed advantages of huge unified systems with something much better for a very low relative cost. These blackboard systems are also much easier to evolve as systems requirements change.

Another huge advantage is that the different vendors are completely free to use any database structure, or database type, that they want. Since databases talk to intermediary databases using the blackboard architecture pattern, the software needs only convert data during the transport between databases, a much more tractable problem than inter-application messaging. These conversion applications can also be provided by third parties.

This model is on its own not sufficient for all interchange[4], but complements the preceding inter-application messaging design nicely.

SICS in Stockholm was working on a database with some of these char-

[4]This is an important take-away: no single mechanism will suffice. You need a mix.

acteristics for India[5]. The open source DIGHT database is intended to be slow synching over high- and low-speed links and be resilient in the face of unreliable links. This kind of database is very suitable to the write-only nature of most medical records data (see page 131), where slow synch does not lead to data consistency problems.

Most obviously, any solution that involves a common database of the "old" kind, and by that I mean a humongous server running a single instance of Oracle or SQLServer is doomed to failure when we talk about a billion patient records as in India. And that is just the current population, more people keep being born...

Which leads us to another uncomfortable truth:

> *Any design that depends on Scandinavian conditions including very few patients, very reliable networks and infrastructure, perfect patient identification, government run purchasing programs, single payer, and pathologically obedient users, is bound to fail in the export market. These conditions of healthcare IT bliss simply do not exist in many other places and more particularly nowhere where there is a lot of people in large need of healthcare.*

On the other hand, if you design for adverse conditions in heavily populated third world countries, you will very possibly have an excellent product for sale even in Sweden.

18.4 Basic interfacing principles

Before going on, we need to look at how computer systems talk to each other. If all systems communicated the same way, we wouldn't have a problem, but we're not so lucky.

There are several independent characteristics that determine information interchange, among which:

- The basic coding of the information.
- The elements that are communicated.
- The assumed states of knowledge of the communicating partners.

[5] I checked in December 2013, and as far as I can see, nothing more has been done the last two years, so the project probably died. Too bad. The link: http://dight.sics.se

Basic coding

The basic coding of the information includes such things as the format of the numbers and strings that are passed along. Are they in binary form, and if so, what character tables are used? As an example we could mention ASCII or Unicode for character encoding.

How are the elements separated and identified in the communication stream? Examples are tagged formats like XML, or separator based formats like CSV (Comma Separated Values).

The HL7 standard[6], for instance, uses "|", "^", and "&" as separators between data fields and parts of data fields. EDIFACT[7] uses ":", and "+" for similar purposes.

Semantics

Semantics describe what things mean, or in the case of information technology, what a code stands for.

Tables of lab codes, or the ICD10 table of diagnostic codes are examples of semantic definitions.

If we send a code "HB" from one application to another, it means nothing unless we also define what code table it refers to. It could mean "Hemoglobin", or it could mean "Horrible Bomb", or anything between, unless we decide what code table both partners agree to use for the interpretation.

Syntax

The syntax[8] of a message refers to which elements need to be present to make sense of it. When talking about communication between applications, "syntax" can refer to "grammar" in the strict sense, or the parameters that need to be present.

For instance, we can stipulate that a message from a lab system to a health care record system must include the patient identification, date and time, which lab did the analysis, plus a series of analysis codes and results. We also need to specify in which order, or in which hierarchical structure, the different elements must appear, so that the receiver can tell which is which.

If everything is present except the patient identification, or if the patient identification is in the wrong place, there is a "syntax error", and the message really makes no sense.

[6] http://en.wikipedia.org/wiki/Health_Level_7
[7] http://en.wikipedia.org/wiki/EDIFACT
[8] http://en.wikipedia.org/wiki/Syntax

Sequence

Many messages presume a certain sequence of messages, or states in the receiving system. For instance, if you send a reply to a lab order, and that reply is identified by a lab order ID, you presume the receiver already knows about that order. Either the order was originated at the receiver's system, or the receiver got another message before this one that defined that ID to the receiver.

If the sequence is defective, such that the receiver does not know what the lab order ID refers to, one of several things can happen:

- The message is rejected with an error.
- The message is saved and set aside until such time that another message arrives that does define the missing data, so that processing may continue.
- The receiver calls out to the sender, or other systems, to try and find out what the ID refers to, so it can fill in the missing knowledge and proceed with processing the lab result.

The treatment of out of sequence messages can also differ depending on what message type we're talking about. In the above example with a lab report, it can be acceptable to let the message through even without a matching order, but if the lab report comes in and refers to a patient ID that the receiver knows nothing about, things become much hairier. A missing patient really is cause for alarm.

In other words, sequencing of messages needs definition, handling, and alarms, according to the type of message and exactly what type of information is out of sequence.

Sources

When two systems communicate, the set of necessary elements in that communication may differ. For instance, to create a patient record in an administrative system, it may be necessary to supply a name, an address, a zip code, city, and a phone number, while to create a patient record in an intensive care unit (ICU) management system, only the name and birthdate may be required.

If the ICU management system needs to send a message to another system, causing a record to be created in the administrative system, there isn't enough data available to do so, which creates a problem. The problem here is a mismatch of information content between source and destination. Any useful message, in this case, must be composed from information from two or more sources at the same time.

The question is what the receiving system should do in this situation. Should it refuse the information? Should it call out to other systems to find the missing parts? If you design interconnections pair-wise, that is single system to single system, you will force both systems to contain *all* the information the other system may need during communications, which is an unnecessary and excessive requirement on such a system.

Standards

Standards tend to solve just a small set of the already mentioned problems. For instance, the basic coding can easily be standardized, and even the semantics and the syntax are easily agreed upon, even though the use of such standards have often failed in reality. Just look at how many semantic standards have been proposed and implemented without much effect on real use. A good example is lab analysis code tables.

But when it comes to standardizing sources and sequences, everything is left to chance and implementers, and for good reason. The type of data a system holds determines the "sources" part, and also determines what the system actually does. In other words, it's an inherent part of the intended functionality of each system and can hardly be standardized without undue limitation on the ability of developers to add useful features.

States and semantics

When we design interfaces, we think of them in two dimensions: semantic content, and successive states.

The semantic content corresponds to the parameters in a call in the Remote Procedure Call (RPC)[9] model, or the attributes of an object, if we're using remote objects. All the parts of one RPC call or all the attributes in a marshalled object[10], are simultaneously present. If one such call is all we need for a full request or response, we're in a state of bliss, otherwise knows as "stateless"[11] calls.

If this one stateless call can be repeated any number of times without causing erroneous results, that call is additionally called an "idempotent" call, which is even more blissful than a plain stateless call.

Tight coupling

If two applications, or two databases, talk directly to each other, we call them "tightly coupled". Any material changes to the interface of one of

[9] http://en.wikipedia.org/wiki/Remote_procedure_call
[10] http://en.wikipedia.org/wiki/Marshalling_(computer_science)
[11] http://en.wikipedia.org/wiki/Stateless_protocol

them must be matched by corresponding changes to the other.

The contents of interfaces are determined in two dimensions which I'll call the "length" and the "breadth" of the interface.

The "breadth" of an interface is all the different elements that are simultaneously communicated across the interface in one call, such as "message identifier", "patient identifier", lab values, etc.

The "length" of the interface is all the different calls that can come in a sequence. Yes, this implies "statefulness", but that's ok, since databases and applications *as a whole* almost always have state.

All of the dimensions

So, in summary, if two systems need to connect directly to each other, they have to agree to all of the following:

- how the data is encoded at a low level
- the elements of data that are communicated together in each transmission block
- the number and order of transmitted blocks

These conditions will only be fulfilled for two systems that are very similar in nature and design, so to adapt any particular system to this interchange will usually involve some serious changes in architecture and code of at least one of the systems involved.

If we introduce a standard that guarantees that two systems can talk to each other, that standard needs to be "maximal", that is include everything that may ever be needed in such a conversation. This implies that not one, but *both* of the communicating systems now need to be re-architected and re-coded to a considerable extent to reach up to the level enforced by the standard. In exchange, of course, this work should only need to be done once, since every system should supposedly be able to use that standard forever after. Until that moment when the next standard appears on the scene and the whole thing starts over, or one of the systems developers can't stand the constraint anymore and just plows ahead with new, delicious, but incompatible feature additions.

A decoupled intermediary

The solution must be to move the message composition and interconnection functionality out of applications into separate processes. This way the individual applications need only import or export the information they

need for their own purposes, and not bother with extra functionality and data elements only needed for communications with other systems.

The external management of the data must be able to receive whatever data applications are able to deliver, in any order they can deliver it, and store the assembled data structures in a database. This system must also be able to actively search out data that is missing from structures, so it can serve up more complete structures to application systems that need them.

Message transformation systems in general can only work with the information available in an incoming message to produce an outbound message, but have no memory of state. They don't keep information from one message to complete another message in the future. This moves the responsibility for the completeness of information back to the communicating applications, exactly the thing we shouldn't do.

My solution to this is an active database based on the blackboard pattern mentioned earlier, which turns out to fulfill all these requirements and also be pretty simple to build.

Chapter 19

What is the difference?

What is the difference between medical records, and say, accounting?

Is it the same thing, except we just need to exchange "customers" for "patients" and "invoices" for "prescriptions"? Nope, it ain't that simple.

There are a number of differences that we need to take into account in order to avoid building completely ridiculous systems. Let's go through a few.

19.1 Different kinds of data

In an accounting program, we have a set number of entities such as "customer", "supplier", "invoice", "payment", etc. Once we've got the entities down, we're not at some later stage going to encounter a new kind of entity we didn't know about, such as a "credit info song in C minor". There is no such thing and there never will be, not as far as accounting is concerned anyway.

In healthcare, though, we are continuously having to handle entities that weren't considered when the system was built. We may be able to handle x-rays, but what if someone comes along with a color x-ray[1]? Or an ultrasound in stereo[2]? Or an interactive microbiology report[3]? Just look at how many EHR systems there are that can't handle digital images, and you'll see what I mean.

Relational databases are excellent for applications with well-known, well structured data, such as accounting programs. Every record consists of a

[1] These do exist, at least they did 20 years ago or so. Color was used to extend the dynamic range of film.

[2] Why not? I made my own stereo stethoscope out of two regular ones once, and it's much better, especially for listening to the thorax of small children.

[3] With smell! Gangrene smell really gets your attention, so transmitting it through the application could be neat.

set of columns, predetermined to be of a certain kind and size. If a thing has a name, there's a column for that. If it has a price, there's another column for that. But if the thing you're trying to save has a value that you didn't provide a column for, it doesn't work, and you can't add that thing to your relational database, at least not without cheating.

In short, the major part of medical records data is not well suited to relational databases. The demographic data, lab data, and most of what happens in operating theaters and intensive care units does fit the relational database concept. Yes, you *can* get anything into a relational database, but you are not exploiting their advantage if you dump formless blobs of binary info into extra wide single columns[4]. You're just faking it.

> *Try to avoid squeezing the data into a mold that isn't suitable*[5]. *Delegate the choice of database technology down to design and implementation, properly encapsulated, where it belongs. Don't make it a primary attribute of your EHR system.*

19.2 Indeterminate communication partners

An accounting system often communicates with other systems, but those systems are known ahead of time and rarely change. An accountant receives data files from his customers, but not from anyone else. A bank expects to talk to its customers and partner banks, not to just anyone at a random point in time.

In EHR systems, however, you may exchange data with the same small group of partners most of the time, but sometimes you need to exchange data with almost anyone in the world. There is no way you can usefully limit yourself to a preselected set of communication partners.

> *Avoid assuming commonality of standards with your communication partners. Again, use encapsulation and delegation to your advantage.*

19.3 Eternal data

In an accounting program, you don't have to be immoral or rude to let old data die. If you do a major system upgrade, it's totally acceptable to dump old data and only keep the end of year balances which are fairly easy to bring over to a new system.

---

[4] You know who you are.

[5] The term in this case is "object-relational impedance mismatch".

In EHR systems, however, there is no "end of year balance", so all old data should be preserved more or less in its entirety when system upgrades or system replacements are done. The time period you have to keep data is open for interpretation and will differ from one country to another, but count on it being somewhere in the range of five to 100 years.

> *Electronic Healthcare Records data must be preserved across system upgrades.*

There is a complication or two hiding here. Do we upgrade all data every time, even unused data? If so, we'll have an exponentially growing data volume to handle. If we don't, we must maintain old software versions up to a century[6] just to be able to read old data. And even then, this is presuming that all data is in one place, which it won't be. There will be offline data, data on USB sticks, on (temporarily or permanently) disconnected machines, and so on.

19.4 Write-only data

In healthcare, data doesn't change, while in accounting data is usually changeable, even though I know you expected me to say the exact opposite. You're not supposed to modify general ledger data but anything not actually posted should be entirely modifiable[7].

Healthcare data, however, is probably always best regarded as write only. Every change should be a new document or record that refers to the document it replaces, without actually removing the preceding document from the EHR. An example could help illustrate this:

Imagine you get an Hb value of 6 from the lab, so you give the patient a unit of blood. Slightly later, the lab updates the value to 10 because they reran the test and came up with a different value. Now, if the original value of 6 is made to disappear and is replaced by the new better value of 10, the doctor who ordered the unit of blood is going to look mighty silly afterwards, and boy, do we hate looking silly.

All information that can have formed the basis for a clinical decision, even if that information is defective, must be preserved in the EHR. It should be made clear by the system, however, that other information has superseded it.

---

[6] Highly unlikely to work. How much digital data have you seen that is 100 years old and still is readable? (Yes, I know, computers weren't yet invented... it was a joke.)

[7] I'd argue for having everything in accounting modifiable and I've built accounting systems like that. But we can't go into that here; it's an entire subject in its own right.

Chapter 20

The perils of user groups

In most areas of IT design, the use of user-groups, or "focus groups", to determine the features and design of future software, has mostly been abandoned. Most requirement engineers have come to realize that you can't ask a future user how your software should work, and expect a useful and correct answer. The average user simply cannot define what he wants, since he doesn't know it until it is presented to him. As Justice Potter Steward once said about pornography: "I can't define it, but I know it when I see it"[1] (paraphrased slightly).

In the area of medical EHR systems, however, user-groups are used more than ever, even though almost everyone involved simply must have realized that this system produces poor, if any, results.

One can only guess what the reasons for this may be, but the following seems plausible:

- By arranging for membership in a user-group, one can give the participating doctors and nurses a (false) impression of having a say in the upcoming system design or purchase.
- If doctors are part of user-groups that have to approve designs, the supplier and the purchasers involved have a convenient scape-goat if the design turns out to be wrong.
- Having doctors and nurses in a closed room arguing about meaningless details keeps them from getting in the way of the real work of building substandard software.

Whatever the reason may be, we should avoid having such user-groups at all, in the interest of good software design, and in the interest of not

[1] http://en.wikipedia.org/wiki/I_know_it_when_I_see_it

keeping doctors and nurses away from their real work. User groups don't work for other designs, be it hardware or software, so why expect them to work for healthcare applications?

The design should instead be based on what good system architects who understand the problem domain[2] envision as a good design, and then tested in a number of small increments. The users should be used to *verify* designs, not to come up with them in the first place. Unless, of course, they also happen to be experienced system architects; then they should be carefully nurtured and listened to[3].

An important complicating factor is this: doctors and nurses today are not IT-naive. In other words, they already know a substantial amount about the use of IT systems, and this for them forms the background of what is possible to do. This is both expanding and limiting the solution space they're capable of imagining.

They're secretly thinking of how neat it would be to have a "Facebook for doctors and patients", then come up with complicated designs that indeed look like Facebook on LSD. Since these doctors and nurses don't know what is technically possible, they are limiting themselves to designs that don't use the full spectrum of possibilities, either in architecture, data models, or user interfaces, on the one hand, and try to squeeze in any neat features they've seen elsewhere, but which don't really fit in, on the other hand.

Another problem arises due to the fact that most healthcare workers are already using crappy software, software that form only part of the solution that is needed. The parts that are missing are a daily and severe problem to these workers, and the need to solve these problems takes precedence over everything else. What they don't realize is that there is a substantial number of working solutions in the current crappy system that they take for granted, but forget to take into account when designing new systems. This easily results in systems that solve exactly the problems the previous system failed at, while failing at exactly the problems the previous system had a good solution for. Coupled with the tendency of the purchasing organization to replace entire systems, and not improve on the constituent parts only, this leads to a fantastic progression of major system replacements at staggering cost, which makes nobody but the vendors happy, and which leads to a comic cavalcade of failed projects[4] that in no way help doctors or nurses to concentrate on their real work.

It really takes someone knowing both IT design and medical practice intimately to detect these gaps in requirements and design. If such a person leads a user group, and totally dominates it, the result could still work. But

[2] Yes, I am one of them.

[3] Yes, I am one of those, too.

[4] However failed these projects are, they're always declared to be a success anyway, for political reasons. See the section on politics of purchasing (chapter 21).

the other members of that group will only serve as drag on the leader, and they're better employed elsewhere. If you're lucky, the group disintegrates and only the leader remains.

If you're really lucky, you have more than one such dual domain person in your team, and then you can form a group with only these people. But that group should not be called a user group, it should be called a design group, and should be put in charge of the design, not kept in an advisory role.

I'm also including a discussion of user groups from a slightly different perspective in the longer discussion on the politics of purchasing (chapter 21).

Chapter 21

The politics of purchasing

Some of the major, if not *the* major, determinants of the (lack of) quality of EHR systems, are the political ingredients of the purchasing and maintaining process. There are two main directions this could take, depending on country or local regulations.

1. EHR systems are selected and paid for by the same group of medical professionals that will use them. In this case, the problems of correct definition and maintenance are similar to those of any other type of system, so I won't go into details about this, since the political factor is less prominent.

2. EHR systems are selected and paid for by one group of people, usually civil servants or political appointees, but used by another independent group, consisting of medical professionals.

The latter system dominates in at least Sweden and the UK, and is the subject of what I'm describing below. Since I have personal experience of only the Swedish system, I'll stick to that[1].

21.1 The purchasing organisation

In Sweden, healthcare is run by 21 separate county councils ("landsting"). These are run by elected politicians, and their main responsibility is healthcare.

Practically all healthcare organizations are owned by their respective county councils, and the few privately run healthcare centers and hospitals

[1] I imagine the UK isn't any better than Sweden when it comes to politicized processes like this one. But that's just my imagination, reinforced by impressions from the TV series "Yes, Minister".

work under contracts with their county council at predetermined rates and conditions. As part of these contracts, these private entities are often forced to use the same EHR system the county uses, including machines, software, and help desk. From the standpoint of this discussion on the purchasing of EHR systems, this makes them indistinguishable from those providers that are owned outright by the county.

A very few entirely private healthcare providers only work with private insurance companies, but their number is totally insignificant, so I leave them out of this.

Purchasing of medical supplies, including IT systems, is done by the county council elected officials and civil servants. The support organizations are also run by the same council, directly or indirectly.

> *So, basically, the IT systems for clinical use are specified and purchased by a political organization with no immediate stake in the system's ultimate usefulness for patient care.*

This also means that the main emphasis of the specified system will be how well it supports the *management* of healthcare, not the day-to-day *provision* of healthcare, under the assumption (a natural one for a manager) that the accurate measurement and management of the process is the aspect that has the most impact on the efficient and effective provision of care[2].

21.2 The manager's viewpoint

The manager naturally sees the accurate estimation of resource use and resulting effect on the patient population as the most important factor in providing effective care. After all, if you don't know what is happening, how can you hope to improve the process? And I have no problem with this, since it's correct up to a point.

The manager, however, sees this in isolation, being too far removed from "the factory floor", and seems to have a tendency to assume this is not only a very important aspect, but maybe the most important aspect, *to the exclusion of clinical needs*. Indeed, since he has no real knowledge of the clinical process, he can't, however much he'd like to, take the mechanics of the clinical process into account when specifying the system.

[2] I don't believe that's true. Quite the opposite; we're measuring the living daylights out of healthcare with comparatively very little provable benefit to actual patients. In many cases, we have a provable *negative* effect on healthcare due to this measurement craze. Additionally, if departments are funded based on scores on these measurements, civil servants without any medical training are in effect directing healthcare according to their measurements, which explains the random walk we often see in healthcare provision.

The result is a system specified to provide as much insight as possible into the flows of patients and staff in providing healthcare at a large scale. From the manager's standpoint, the healthcare staff serves mainly as data entry clerks to feed the machine enough data to get a clear picture of the organization as a whole.

> *The manager assumes he is also providing for the needs of the healthcare staff by creating a system that organizes* all *the information about the patient that can be found. The basic assumption here is that the more information we have, the better the care will be, which is not unreasonable for a lay person to deduce, since so much of the complaints of healthcare staff center on missing or mislaid information. The problem with this deduction is that the most frequently heard complaint not necessarily indicates the most effective solution. Additionally, this deduction is plain wrong.*

21.3 The user's viewpoint

The clinical user doesn't come into contact with the management aspects of the system, so is often left scratching his head when it comes to figuring out what the system is actually for. A lot of data entry seems unrelated to the healthcare process, and even downright obstructive. To this user, the system seems to demand more and more attention while not giving back much of direct usefulness.

Interestingly, the quickly increasing mass of information about each patient turns out to be counterproductive. If the information isn't structured right, and it isn't, it only becomes a massive pile of data to plow through to find the few truly useful pieces of information in there.

Users of these large data hogging systems increasingly avoid reading through the massive prose and turn to the patient in the hope that the patient himself will be able to give his story in a nutshell. What we often see is thus that today's huge EHR systems ultimately provide *less* utility to the healthcare worker than the more limited systems of the previous generation.

21.4 The user group

I've discussed the user group before (chapter 20), but there is more to be said about it, in particular in the context of the purchasing process.

There are two ways the actual users are commonly involved in defining systems to be purchased and that is either as advisors to the purchasers

in the form of a user group, *or* as members of the purchasing organization itself, but with a background in direct healthcare, usually as a doctor or nurse.

If the advise is gathered from a user group, all the usual problems of these groups occur:

- People assume a level of common functionality which they don't think is necessary to express formally, but which is foreign and unknown to the lay people receiving the advise.
- The participants each have their own agendas, largely based on their own major frustrations with other systems, skewing the requirements gathering into a list of aggravations rather than a useful description of a system.
- Many of the items will be "false requirements" (chapter 15), i.e. solutions expressed as requirements.
- Very little of the requirements will relate to the needs of the administration, making it too easy for managers to dismiss the entire list of requirements as useless, and the users in the group as "out of touch".

Another major problem is that the user group participants set themselves up to become "fall guys" if the requirements turn out to be a problem. Since they are the weakest group, the group with the least political and organizational power in this whole setup, any problems that occur will be at least partly blamed on poor requirements, and thus the users themselves. The wily and experienced user therefore avoids being part of the user group, leaving it up to the young and inexperienced users, which in turn diminishes the value of the requirements, and increases the subsequent cynicism in users. Nothing good comes of this.

If, on the other hand, the clinical user is transplanted from clinical work to become part of the purchasing organization, he far too easily and far too often soaks up the attitude and priorities of the organization he joined, losing all effectiveness as an advocate for the interest of the clinical workers. It takes an extremely exceptional person to withstand this peer pressure and maintain a clinical attitude under these circumstances, and such persons are far too rare. Worse, these independent characters are very unlikely to let themselves be moved from the clinic to the administration, unless they are offered a very high position. And if they are, they'll end up too high in the hierarchy to influence the actual system requirements, anyway.

Summary
2014-04-01, 15:10:05

Differences exist between documents.

New Document:	Old Document:
draft_10	draft_9
118 pages (6.30 MB)	112 pages (4.61 MB)
2014-04-01, 15:09:33	2014-04-01, 15:09:30
Used to display results.	

Get started: first change is on page 1.

No pages were deleted

How to read this report

Highlight indicates a change.
~~Deleted~~ indicates deleted content.
 indicates pages were changed.
 indicates pages were moved.

21.5 The support organization

Once the system has been specified and purchased, then delivered in some form, it needs to be rolled out to the users, and a user support organization needs to be set up.

The management of this rollout and support organization will ultimately be the same people who largely specified and ordered the system, and this causes a lot of grief. What happens when things go wrong? What do the responsible people point to when the system doesn't work for the users? We have to separate the problem space into problems that the management users have and problems that the clinical users have, since these two groups are, as far as the rollout and maintenance is concerned, in non-aligned roles.

Management users' problems

Management users, as all users, will experience problems with any new IT system product. Whatever the problem, be it interface problems, missing functionality, or breaking defects, these users usually will contact the vendor of the software either directly or through their own support group to get the problem fixed. There is nobody between them and the vendor except other people from their own organization, so there is no resulting conflict of interest. It's all pretty much straightforward. They usually get their needs met.

Clinical workers' problems

Now, clinical workers' problem resolution is a much more convoluted process. These users are usually prohibited by management from contacting the vendor directly, since neither the management organization nor the vendor want that to happen for a number of reasons[3].

When the clinical staff, doctors and nurses, turn to the management organization, or its support desk, with complaints, the first step the support staff takes is to try to resolve trivial problems. If that doesn't succeed and the problem seems to be caused by real software bugs, the problem is forwarded to the vendor for resolution. If it turns out that the problem is caused by defects in the design itself, such as missing or wrong functionality, the support organization usually responds with denial, claiming it's the user

[3]One of the most important reasons is that users are expected to ask for too much work to be done. Now, one can argue, as one does, that it's wrong for the management organization to select which requirements are acceptable to farm out for work. But management clearly does not agree.

who doesn't understand the system. If this "resolution" doesn't suffice, the next step is finding someone to blame[4]. Let's see how that works out:

- You could blame the people responsible for specifying and purchasing the system, but that would imply that the management organization must point to itself as the problem, and that will not be their preferred choice[5]. Remember, the support group is part of the same organization as the purchasers, often sharing an office and knowing each other pretty well personally.

- You could blame the vendor for creating a poor implementation of an otherwise well specified system, but that won't fly since most vendors have taken care to be fully covered contractually against any such shenanigans. Sometimes, however, vendors can be induced to take the blame if compensated in other ways, with other contracts or payments, less obvious to public scrutiny. But that compensation needs to be significant.

- You could blame the user group that was responsible for the requirements process, and this usually works out just fine for the organization as a whole, in particular since the livelihood of the user group participants doesn't depend on user groups, which lets these people go off in a huff and swear never to help out again. And that suits the management people just fine, since they'll also be both discredited and disillusioned enough never to push for better management and IT systems again. A real win-win situation.

- You could blame the users for being anti-technology or plain stupid, which also usually works out fine. If you paint your best doctors as being ignorant when it comes to computers, people in general will have no difficulty accepting that. It fits a narrative that smart people must have other areas in which they're really dumb, and this idea is easily "sold" to outsiders.

After all is said and done, *someone* gets the blame, but the problem itself rarely gets solved this way. Somewhere down the line, however, when the whole episode is almost forgotten, the problem gets fixed in an update. Or maybe it never does.

[4] This is, after all, a political organization, where affixing blame is often as important, or more important, than making the right decisions in the first place.

[5] I've never seen that happen. One could almost get the impression they never make mistakes. Amazing.

21.6 When the shit hits the fan

When you consider the factors I already summed up, it becomes fairly easy to see what's going to happen once serious problems show up in the practical use of new IT systems in healthcare[6].

When the users report with steadily rising concern, confusion, and anger that a particular system either isn't usable for its purpose or takes far too much time from other work because it's a Rube Goldberg contraption[7], the management initially tries to ignore it all, claiming and hoping that the users will either find workarounds, get used to it, or both.

As the problem becomes so large that it can't be ignored anymore, the support organization has a discussion with the vendor, hoping the vendor will fall on his sword and admit responsibility for the cock-up, but that rarely works. The contract almost certainly clearly stipulates that the purchaser is responsible for the requirements and the feature set, and as long as the features delivered can possibly, even by a long stretch, seem to have some relationship to the requirements, the vendor is in the clear. This is a lesson most vendors learn early on, that money and time spent on watertight contracts really pays off, or they won't survive contact with political organizations for very long.

Since the vendor now has gotten away without taking hits, the blame rays turn towards the user group that collected the requirements and were supposed to advise the purchasers as to what to specify. This group is just way too easy to assign blame to, since the group members themselves where never quite interested enough to keep careful notes, nor politically savvy enough to cover their asses while there was still time to do so, nor technically knowledgable enough to call bullshit on the arguments. They're also the group that can survive criticism like this most easily, since it never was their job to do requirements engineering in the first place. They were, at best, just happy amateurs with an idealistic agenda. At this point, blame is squarely assigned to the user group, but that doesn't solve the problem. Users still can't work with the system, and they're still making undesirable noises. Questions start to rise not only about the user group, but why the organization relied on that user group in the first place (which, in fact, is a very good question). More blame, with a hint of a resolution strategy, needs to be assigned.

The turn now comes to the users who raised the complaint. It's worth trying to tell them they're "doing it wrong". If blame can be fixed on them, you achieve multiple goals:

[6] It also helps to have read Machiavelli: The Prince.

[7] Rube Goldberg invented a number of machines doing simple tasks in insanely complicated ways, just for the fun of it. That nicely describes our current EHR systems, bar the fun.

- You've find a real specific person to blame, and it's the same person that complains. It's all his or her own fault. Recursive bliss.
- You're making the complainers shut up, since you've demonstrated that the ones that complain are the *cause* of the problem. They're only outing themselves. Silence ensues.
- You're diverting attention away from your own deficiencies and poor understanding of the problem domain.
- You can present a solution: train the users more, since they're clearly underperforming and not understanding modern technology. Or, even better, not *willing* to use modern[8] technology.

And yes, with sufficient training and effort you *can* work around some of the deficiencies in the system, but it will always be more expensive and less effective than if you corrected the deficiency in the first place. But if the training doesn't help, you can use that as proof that the users are *really* stupid. You simply can't lose.

21.7 So, how do you fix this?

The really fundamental problem here consists of two parts:

- The system is designed for management, not the healthcare process as used by doctors and nurses. The system will *never* be primarily designed for clinical use as long as the clinical users are not put in charge.
- The same organization is specifying and purchasing, then supporting the system, and assigning blame.

The organization that makes the important decisions about the new systems, what it will consist of, how it will work, the price and payments for deliveries, the rollout, the training, and the support organization, is at the same time the organization that will have to resolve any problems that may result from this process. We're figuratively putting the fox in charge of the hen house.

What we *must* do is reform the political and civil service organization that is in charge of the specifications, purchasing, and maintenance of these large systems, to achieve a separation of responsibilities. The group that

[8] For varying quantities of "modern". In many of these cases, the EHR designs are more Flintstone-like than "modern".

makes the decisions for definition and purchase must *not* be the same organization that then needs to resolve any problems caused by defects in the specification and purchasing process. You can't expect the people in this organization to blame themselves when things go wrong.

In order to get this done, we need an awareness of the problem by elected officials in charge of reorganizations at this level. We also need to find a way to make these elected officials willing to take it on, and here it becomes more difficult. This must involve quite a bit of lobbying, finding the right pressure groups, and responsible journalists, all tasks I'm utterly useless at. I can tell you that you have to do it, but I can't tell you how. That's your job.

Chapter 22

OpenEHR

There is no avoiding a little discussion about OpenEHR[1], which is an initiative to structure the medical record. In the preceding text, I've ignored it completely, just as I've ignored a lot of other initiatives. My rationale for ignoring them is that they are all failing my sniff-test; they're all awfully complicated, and they all require doctors to work in a way I (and perhaps other doctors) regard as unnatural. They all seem to enlist doctors and nurses to become input clerks to the machine.

Now, OpenEHR has many of those drawbacks, but since it's the result of a large group of people thinking about this, I can't ignore it. On the other hand, the more people involved, the greater the complexity and the less chance there is of the result being correct and useful. And that is what I think OpenEHR illustrates more than anything else.

The basic problem with OpenEHR seems to be that it's an attempt to bring structure to the current note taking process in healthcare, which by necessity involves limiting and structuring the vocabulary we use to take notes. In other words, it forces humans to produce output only a computer would be happy reading. It adapts the humans to the machine instead of the other way around.

As if that wasn't enough of a problem, the next major problem is that even *if* we could produce input to the machine that is syntactically and semantically perfect, there seems no point to it. What, exactly, will the machine *do* with the information it now understands? According to the goals stated by the OpenEHR organization, it would improve the *interoperability and computability in e-health*, but it doesn't mention what this means or how this would improve healthcare. Now *that* is one hell of a problem.

From their homepage, I took this sentence:

[1] http://www.openehr.org

> The essential outcome is systems and tools for computing with health information at a semantic level, thus enabling true analytic functions like decision support, and research querying.

"Decision support" is vague enough so I can't shoot it down on sight, but "research querying" is a really bad idea. See my earlier chapter on why (chapter 13).

Part IV

A consistent design

Summary

In the foregoing, I've discussed what the problems in healthcare are in general, and how current EHR systems attempt, but fail, to solve them. I've pointed out a number of ways these systems are badly conceived and fail at their task, while also trying to point to any reasons I see why things have turned out this way. I've also discussed the politics of purchasing, a not insignificant problem.

In what follows, I'll try instead to reason out what we need, and how we could go about achieving that. It's clear that we need more than one thing, and that there are a number of aspects of healthcare, that each could use its own solutions.

While discussing this, we also have to discuss how the different solutions interact with each other. Sometimes solutions make opposite demands on the system we're building, so we have to compromise. Let's roll.

When we work with software applications in general, there are distinct aspects of the application we can and should discuss in isolation from each other[2]*.*

One such aspect is the composition of the data, what form the knowledge about the patient takes. Is it numerical or textual? Is it structured hierarchically, or by references, or relationally, or a mix of these patterns?

Another aspect is how this data about the patient is presented to the user, all the different ways this can be done, and which of these ways is useful for which kind of data. This is also influenced by the environment and situation of the user, so much so that the presentation could be radically different between, say, an operating room and a ambulatory setting, even though the data is the same. Think about how an x-ray is presented, for instance. In the OR, it should be presented graphically, preferably next to other graphic material such as NMR or ultrasound images, while in the ambulatory setting, a simple textual protocol could suffice.

Finally, the operations that should be performed on the data differs. For some data, duration and intensity could be important, while for other data the change over time is crucial.

[2]In this discussion, the software engineers among the readers will recognize the Model-View-Controller pattern, and a bit of Presentation-Business-Data layer abstraction. Good for you.

Chapter 23

The phases of the clinical process

In what follows, I'll view the different steps and actions in the clinical process one by one. For each step, I'll first describe the step as such, independent of any IT systems. Then I'll describe how current EHR systems help or hinder this step, and finally I'll describe what future systems could and should do.

23.1 Clinical encounter

The encounter can be an ambulatory visit to the doctor, a phone call, a five minute administrative time set aside to review reports for a patient. It could also be a visit at the patient's bed during rounds. There isn't that much to discuss about the encounter as such, since it mainly consists of a coming together of doctor, patient, at a certain place and time. The encounter could be said to consist of the elements described in the following sections.

How it is

In current systems, the encounter is the main element in the medical record, as far as the notes are concerned. It is very clear what was done during an encounter[1], but it's much harder to find what was done in relation to a particular issue if you don't know when it was done or by whom.

The encounter is also clearly related to a doctor or nurse and a place, so if you're mainly interested in that information, it's there. From a medical standpoint, however, this information is practically useless.

[1] "Clear" for arbitrary values of "clarity". What is often not clear are which orders were created or prescriptions written.

Some systems record the creation of referrals and prescriptions right in the notes, but even then, they rarely link directly to the created documents. In other systems, that information isn't in the notes, so it is left to the user to look for documents with the same date and time as relevant notes to try to piece together what really happened[2].

Even if this all works fine, which it doesn't, it helps very little in finding the notes relating to a particular issue, which is really all that usually interests us as doctors.

How it should be

The clinical encounters should still be ordered chronologically in future systems, but that is a view that will rarely be used. A better organization is to have notes, documents, results, and prescriptions, ordered according to issue. If I'm opening up the records for a particular patient, the very first thing I should see is a list of issues, such as "diabetes", "hypertension", "headaches", and so on. If I select one of those issues, I should see a list of notes, documents, results, and prescriptions related to the selected issue.

If I, on the other hand, select a document, a note, or a prescription, I should see a list of issues related to that information element, and through that list of issues, I should again be able to see related information elements, as described in the preceding paragraph.

23.2 Overview of patient history

As a doctor seeing the patient, the main things we want to know about the patient and the encounter are:

- The main problems the patient has or has had over the years.
- Which problem or problems we as doctors need to consider and manage today.
- Which other problems are of importance when considering the current encounter-related problems.

[2] As if that horrible state of affairs wasn't enough, at least one system I know has a tendency to show the wrong date and time in either or both of those places (notes and documents), making it well nigh impossible to figure out how they relate to each other. That same system, by the way, also records users differently in medication lists and notes, the former with a user login short code, the latter with the full name. In neither place can you see both forms of user identification, so it is *impossible* to match up notes and prescriptions. That reflects some *amazing* dedication to screwing things up.

- What has been done and decided so far concerning our encounter-related problems.
- What has happened that isn't in the record but is relevant[3]. These are the things the patient should be able to tell us about.

How it is

In current systems, there is nothing but the "history" and the "assessment" fields in the notes to reconstruct the patient's history, unless you count going through all referrals and prescriptions and trying to deduct from those what could have initiated them.

2014-01-10	**Martin Wehlou, Doctor's notes**
Contact	New visit
Reason	Shortness of breath
History	Had similar complaints in 2011, got Salbuterol and improved on that. Last spring also got Pulmicort. Patient still uses both, but is still more affected in cold weather. Pulmicort 200: 1+1 doses per day. Uses Salbuterol every now and then, several times daily. Patient feels pressure over the chest during effort.
Status	BP: 145/85, sitting, left Cor: regular rythm, no souffles Pulm: ves. sounds bilat. ECG in rest: normal. PEF: 225
Assessment	Needs more Pulmicort. I'll order treadmill ECG. Also giving patient an electronic PEF meter for a couple of weeks to collect everyday data.
Referral	Referral sent to UAS, Physiology lab
Prescriptions	Pulmicort Turbohaler 200 micrograms/dose 3x60 doses, renew x2

2014-01-16	**Martin Wehlou, Doctor's notes**
Contact	Followup

Figure 23.1: A typical entry in a classic EHR

History fields are present in each encounter note, so gathering this information implies that you have to scan through all the notes, or all the notes you can stomach, and mentally assemble it all into a consistent whole. If the record systems contains notes from a number of different departments, you can choose to go through them separately or all together, if the system gives you that choice. Assuming you're allowed to scan through the notes for other departments without specific authorization and/or reason.[4]

[3] Note that I carefully avoid saying "since the previous encounter", since all too often even important events that have happened *before* the previous encounter are not in the record, and only the patient can inform us of them.

[4] Since we built up a picture of the patient's history by scanning the notes, it's obvious that we don't really know what we're looking for until we've found it. So we often can't have a known reason for scanning a particular set of records, even though the reason for doing it would be clear if and when we could see those records. Interesting circular

During most encounters, we simply skip this step, hoping we didn't miss anything important, but sometimes you painstakingly have to assemble a consistent story for some particular purpose, like a referral or a report to the health insurance, and then it's convenient to copy that fairly comprehensive history into the record for later use. The problem, however, is that there is no place to put it except inside the current note, which means that this very useful summary slowly recedes over the visible horizon as later notes accumulate. The only way to keep it current is to keep copying it over to more recent notes, and updating it as you go along, but this is, even at best, a really ugly solution. I've heard from other users that some systems even have a function to copy old history notes over to the current note, but that's a really terrible idea, since it in reality means a large amount of duplicated information, which (the same user reported) doesn't get updated anyway. It simply results in outdated information being added under a current date. A blatant lie, then.

How it should be

If there is a list of issues, aka health problems, over the years, this in itself constitutes most of what we're looking for in a "patient history". The next level down in detail can be found by opening any particular issue and seeing only the notes and documents referring to that one issue.

For some issues, a timeline can be a good visualization. For example, the long term parameters and therapy of diabetes have something to gain from a well-designed timeline presentation. Other examples are recurring ear infections in children, or vaccinations, or the followup of pregnancies. But there are also a lot of examples of issues where a timeline doesn't make much sense, such as surgery for hemorrhoids or ingrown nails, or a pneumonia. Since the utility of timelines is so heavily dependent on exactly *which* issue we're talking about, it should be a part of the issue template set for those issues. This has the added beneficial characteristic that the timeline can be designed optimally for each issue type.

There is also the possibility that we'd like to see *all* issues in an overview in the form of a timeline, which could have some utility. In this case, the timeline will be much simpler and not convey much information about each particular issue. Some issues will not be there at all, not in a useful form at least. Just think of congenital diseases like sickle-cell anemia. Yes, each complication of sickle-cell anemia may occur there, but the sickle-cell trait itself can't usefully be represented.

---

reasoning going on here, and the reason for *that* is that the record is oblivious to the concept of "disease" or problem, so it can't present any information in a cohesive and relevant frame.

For some issues, an anatomical overview could also be useful, such as for a multiple trauma, where damaged areas and organs can be highlighted. In rheumatic polyarthritis, an anatomical drawing with each affected articulation highlighted can also be very useful. But most issues won't lend themselves to an anatomical depiction. Think of schizophrenia, for instance. Would you highlight the brain for that? And what information would that really convey? The anatomical view could be useful, but should then also be designed for those very issues where it *is* useful, not for the entire patient history of problems.

23.3 Clinical examination

The clinical examination consists of a number of manual actions to check on signs and findings. Examples are:

- Listening to heart sounds.
- Listening to lung sounds.
- Palpating the abdomen for masses, tenderness, percussion[5].
- Examining ear drums, eyes, throat, etc.
- Measuring blood pressure.
- Checking reflexes.

Exactly which clinical examinations we do depends on what we're looking for. If the patient presents with an upper respiratory infection, the examination is directed towards those elements. If the patient comes for a yearly checkup of diabetes, we have another set of clinical examinations we ought to perform.

Every doctor knows how to do the most common examinations like using a stethoscope to listen to heart sounds, palpation of the abdomen, and ear inspection, but other examinations are harder to remember how to do. Examinations for shoulder problems or nervous system diseases can be quite intricate and for these the doctor may need reference material to remind him of which examinations he should do, and exactly how to do them and what they mean.

[5]Percussion: tapping and listening to the sound. This can tell you the size of the liver, fluid or air in the intestines, and more.

How it is

In current systems we usually have only a template containing a list of keywords. In some systems you can add keywords or sets of keywords on the fly. If you're seeing a patient with diabetes for a yearly checkup, you'd add the set of keywords for just that. But notice that even if you add that set for diabetes, the system doesn't really take note that you're doing an encounter for diabetes, and doesn't save that fact in any useful way, even though that information is clearly available from the user.

Other systems (or often the same systems, but configured differently) instead have a single or a very few templates with keywords. Since you can't easily add in keywords on the fly, these templates tend to be huge, and to include all kinds of keywords that are only seldom used. Often only one in ten keywords are used during an encounter, so much so that the hunting for the right keyword to enter information becomes a significant time consuming task in its own right.

Both kinds of systems usually only save keywords that do contain entered information into the notes, avoiding saving long lists of empty keywords.

As to how information is entered into the keyword data field, there is very little attention paid to that. Most fields are plain text, with a few fields having a numeric type, which in general is more hindrance than help. Some systems allow for graphing of values over time if the field is numeric, but this has very limited utility since it's done for so few fields. At times it could be useful, such as showing a graph of HbA1c values over time for a diabetic, but this has then to be done by switching over to the lab results module, then picking out the HbA1c value and graph it from there. It's not a part of the clinical overview for diabetes, simply because no such overview exists.

Another aspect of the use of the IT system in conjunction with the clinical examination is the office layout. A physician's office usually contains a desk with the computer on it, a chair for the patient, and an examining table. These can be arranged such that the computer can be used while at the desk, or easily reached while examining the patient on the table or gurney. But in some offices, the examining room is physically separate from the office, and there is no computer in the examining room.

If the examination is done in a room without a computer, the doctor is back to either memorizing his findings or taking notes, the exact workflow we try to eliminate. If the examination room has a computer, we usually can't log in to the same EHR system from two places with the same user, either because that wasn't a part of the design criteria, or because we need a smart card to log in, and it can only be used in one machine at a time.

How it should be

Since the choice of suitable keywords depends on the healthcare issue we're seeing the patient for, it becomes natural to connect the issue to a list of keywords. Since the national registry, and communicable diseases reporting *also* depend on the healthcare issue, it is obvious that those keywords should also be determined by the issue.

Regardless of the reason the user defines the issue, be it in order to get the keyword list right, or to do reporting, or to maintain a correct patient overview, the end result is the same: a descriptive issue in the history overview *and* the right keywords for the clinical notes and the reporting.

> *The "healthcare issue" is such an obvious integrating element that it is hard to conceive how systems continue to be built without that element in their design.*

Not only should the set of keywords (or "items" as I prefer to call them) be determined by the issue, but the contents the user enters as values into items should be determined by the item. The user should almost always be presented with a default normal entry value, a series of common alternate values, and the option to enter free text. This not only speeds up entry of normal values, but also mildly suggests standard values such that the record becomes more easily exported for reporting purposes, and automatic translation to other languages and coding systems.

The user should never be limited to predetermined choices, only invited to use them. Whenever he bypasses them and enters free text instead, that effort will always be greater than choosing a preexisting alternative, clearly signaling a defect in the list of available codes. Any such manual workaround of the coding system must be caught and lead to updating of the coding system to fill in the holes. See the discussion on coding systems (chapter 32).

If the examination room is separate from the office, it has to be provided with a desktop computer mirroring the one in the office. Only one of the machines should be active at any point in time, and the ideal way of achieving this would either be by smart card, which when removed just freezes the screen and keyboard, not logs out the user. Alternately, a proximity activated system using NFC cards or similar could work.

When designing this, keep in mind that the exam room may be shared by several doctors, so when a doctor activates the desktop, it needs to show the same active session that particular doctor had available moments earlier in his office.

Another solution entirely could be a portable unit like an iPad, which the doctor can simply carry back and forth and do all of his work on.

23.4 Creating referrals and orders

Referrals are created when the doctor wants to have another doctor examine the patient or perform needed therapy. For instance, if we have a patient with headaches we can't say with certainty if they're tension headaches or something worse, we may want an x-ray or MRI of the brain, or maybe we want a neurologist to take a look and give an opinion.

In these cases, we start with a problem we can't solve. We have an idea of the general problem area, like "headache" in this example, and what kind of doctor is a specialist in that.

The next step is to find out where there is a neurologist in our geographical or administrative area we can send the patient to. If there are several, we may want to select them according to the patient's preferences or which one has the shortest waiting times.

Once we decide on which neurologist to consult, we write up a referral with the following elements:

- The actual question, such as "headaches of uncertain origin, please advise".
- A short history with major reasons why I can't figure it out, which examinations I've done, and including any results from MRI, lab, etc.
- Related documents in the form of earlier referrals with replies, and documents included with those replies.

After that, we send off the referral using the communication means at our disposal. Sooner or later, the patient will be seen by the neurologist and we'll get a reply back. See below in the section on receiving resultsReceivingResults (section 23.8).

How it is

In current systems, referrals, lab orders, x-ray orders, and creation of general documents such as letters and attestations, are completely separate from the notes and from each other. The creation of notes regarding a particular health-care issue has no influence at all on the document or referral creating process. Even if I include the template keywords for diabetes, nothing in referrals or prescriptions is tuned to diabetes care. Everything is separate and starts from scratch at each use.

If I'm having an encounter with a patient for his yearly diabetes followup and I open up the prescription module, I'm presented with the same choice of thousands of products as always. There is no preselection of products or product classes that are more relevant to my patient. The same thing

happens with referrals and orders. The range of presented choices is totally independent of the actual problem the patient has, and is therefore always excessive.

Each document or referral I need to produce starts out with a forced decision on where to send it. There is no smart preselection of relevant addresses, just a full list of *every* address the system knows of. Also, there is no indication of which addressees are interested or able to handle the problem I wish to present them with, except possibly as indicated by the name of the department. For instance, the "cardiology" department is clearly doing something related to the heart, but do they also take children with heart problems, or do those go to pediatrics? Does the cardiology department do ultrasounds of the heart, or is that the radiology department, or even the clinical physiology department? As a user, I'm clearly supposed to know all these rules and exceptions, but I don't. This is something the system should do for me.

Next, I need to fill in a question (in the case of referrals), and even though I've probably already formulated that in the notes under "assessment" or "planning", there is nothing in the system that makes it easy to reuse that information. Worse, in some systems there isn't even a way of copying that information via the clipboard, necessitating retyping it verbatim.

Finally, I need to rehash the main history of the patient, and the background information I may have. Again, the system won't help me copy over history or assessment notes, or even let me copy over other related documents.

After I send it off, there's a significant risk I'll get the referral back, either because it's sent to the wrong place, or it doesn't carry with it all the information elements the recipient requires. Since these referrals are based on the paper model, the electronic referral now becomes useless, *and has to be written all over again*[6], even if the change is minimal or only the destination address needs changing.

How it should be

Since we're working with an issue, the most probably scenarios for referrals should be already provided in the system, with a brief synopsis of which circumstances warrant referrals, and what elements should be included.

Since the issue template is aware of the reasons for referrals, it will also know where to send them, and what should be put into the "cause" field of the referral. It is also aware[7] of the required extra information that the recipient of the referral will need, so it includes that as well from the other

[6] Accompanied by piercing pig squeals from undersigned, who *still* can't believe how idiotic these systems can be.

[7] Can't help anthropomorphizing the poor computer. I hope it forgives me.

sources in the EHR. The only thing left for the user to do is to verify and possibly add a personal touch to the different fields in the referral, then send it off.

The chances of the referral being sent when necessary, to the right place, for the right reason, and with the right information, will be hugely increased.

If the recipient, against all odds, still sends the referral back due to misaddressing or insufficient information, the user should be able to update it and send it again, to the same or a different recipient, without starting from scratch. We've had that functionality since forever in plain email, so it's clearly not rocket surgery.

23.5 Creating prescriptions

When we institute or extend treatments with medications, we need to create prescriptions. When we start a new prescription, we do that with the following considerations:

- The problem or disease we want to treat determines the therapeutic class of products we want to use.[8]
- We check that there are no contra-indications for use of this therapeutic class for this patient. In other words, does the patient have some other problem that precludes the use of this class of medication?
- We check that the patient isn't already receiving any other medications that may interact with this medication. If so, either adjust dosage of one or both medications, or avoid giving one of them.
- Depending on the patient's weight, and kidney or liver function[9], and which problem or disease we're treating, determine the right dosage and duration of the therapy.
- We locate the information on which commercial preparations of this therapeutic class that this particular hospital or region supports or recommends[10].

[8] To see the whole list of therapeutic classes more formally, see the "Anatomical Therapeutic Chemical" (ATC) classification system: http://www.whocc.no/atc/structure_and_principles/

[9] Many pharmacological products are eliminated by the kidneys or broken down by the liver, so if these have reduced function, the dosage may have to be adjusted accordingly. In some cases, we must do lab tests to determine this, measuring either the liver or kidney function, or the actual concentration of product in blood.

[10] Many healthcare organizations make deals with pharmacological companies to get better pricing. These deals result in lists of "recommended products" that doctors are asked (or required) to follow.

- Finally we get around to writing the prescription.

How it is

In current systems, the prescription module is independent of the rest of the system. Yes, it *is* related to the patient and the prescribing doctor or nurse, and department, but not to the current issue, referrals, results from lab, or anything else that is a concept in healthcare.

This means that when I'm seeing a patient for a middle ear infection, and I want to prescribe an antibiotic, the system presents me with *all* products it is able to prescribe. It is as easy, or as difficult, to prescribe penicillin as it is to prescribe an anti-psychotic medication at this stage. This also means that if I'm not certain of the name of the product I want to prescribe, I have the universe of all possible products to plow through, instead of the relatively limited subset of products relevant to ear infections.

In many systems, you can select products according to ATC grouping, limiting the searchable universe somewhat, but simply finding the right ATC group is largely duplicated work. After all, we just told the system what is ailing the patient, why do I have to keep telling it that in one fashion after another?[11]

Once I've located the product, penicillin V in this case, I need to decide on a dosage. Since the dosage differs according to indication, i.e. it's not the same if the penicillin V is prescribed for ear infections, as it is if it's used for a pneumonia, and still different from the dose for sinusitis. Even worse, none of these dosages are in the EHR system, anyway. So I have to find the reference information on the product, usually through a web site. Most EHR systems connect to a reference site and do look up the product for me at the click of a button, but I have to take it from there, scanning the info, deciding on which table of dosages is applicable, switch back to the EHR and enter the dosage in the form. Then switch back and forth a couple of times to make sure I copied it over correctly.[12]

How it should be

If we have selected "middle ear infection" as the current issue while examining the patient, the issue template will already contain a list of rec-

[11] Yeah, I know, it's rhetorical. It's because the system is idiotic and doesn't have the concept of "issue" or "disease".

[12] And if my boss is stingy, as they practically always are, I'm stuck with a far too small screen with a far too big EHR window, so I can only see *either* the EHR *or* the penicillin V documentation, not both at the same time, hugely increasing cognitive load and risk of errors. This type of dumb savings of a few dollars often get me into fits of rage. I haven't beaten computers down to their component dust, though; I decided that writing this book was a better use of my anger.

ommended therapies, including a few different antibiotic classes, so I can choose either the most likely to work, or the one the patient is known to tolerate.

When I select one of the products in the issue template, it already comes with a recommended dosage for "middle ear infection", since it's a part of that issue template. It can easily calculate from there according to age and weight, or ask for the weight if it is needed and not yet available to the system. The duration of therapy is also dependent on the issue, and can be automatically proposed to the user.

In some cases, including middle ear infections, the duration of the therapy can be influenced by prior diseases. If the infection is a first occurrence, the recommendation could be five days of antibiotics, while if it is a recurrence within a few months of a previous middle ear infection, it is recommended to take the antibiotic for 10 days. If the EHR is using "issues", it can easily detect that a similar issue was active less than, say, two months previously, and then suggest a 10 day therapy to the doctor.

We can go even further. For the sake of argument, let's assume the doctor changes his mind about the diagnosis a couple of days later when the therapy doesn't seem to work. For whatever unlikely reason, he wants to change his diagnosis from "middle ear infection" to "hairline fracture of the skull", after discovering that the patient fell down the stairs and having done an x-ray study. He then deletes the issue "middle ear infection" and replaces it with "skull fracture". As he does that, the system can then ask him if he wants to continue the antibiotics, since the system knows that "middle ear infection" is the indication for the antibiotic. It's aware of the *why* of the prescription.

If the doctor removes the issue "middle ear infection", all the data entered into the corresponding template remains in the EHR, it's only the "framework" of the issue template that is removed. Naturally, both the addition and removal of the template for "middle ear infection" is kept in a history log, but as the diagnosis changes, it doesn't need to remain at the front of the visible record.

23.6 Creating the note record

The note record mainly consists of free-form text. One can argue that this is one of both the best and worst aspects of the medical record. It's one of the best aspects since it allows a fully unbounded and expressive description of the patient's condition, wishes, fears, *and* the doctor's assumptions, vague intuitions, and decisions. It's also one of the worst aspects, since it won't allow the computer to anticipate actions, locate suitable support tools, and warn for missing actions and errors.

The note record usually contains the patient history, the clinical examination, and the doctor's conclusions, and open questions, but in this discussion I'll limit myself to the "subjective" part[13], the "assessment" and the "planning" parts of the note, since the remaining "objective" part is discussed separately as "clinical examination" (section 23.3) and "results" (section 23.7).

How it is

The "subjective" ("history") and "assessment" parts of the note are currently free text only, which is really all they should be. The problem is mainly that they're mixed in with the clinical examination, results, and planning parts, which they shouldn't be. Taken together, this creates a mixed bag of free text, and text that really should be more structured.

In many cases, systems structure a few clinical examination fields with drop downs or numerical masks, but this doesn't really make any significant difference, while only being irritating in the cases we need to add something that doesn't fit the preconceived notions the developer had of valid values.

How it should be

I would preserve both the "subjective" and the "assessment" parts of the record as unstructured text. Humans are unparalleled at describing exactly what is perceived, including the degree of uncertainty and vagueness. This description of impressions, state of well-being, and intuitions is very valuable and should not be hindered by unneeded structuring.

When attempts are made to force the user into using a stricter and more well-defined language in these parts of the record, we force the human user to do the work machines are better at, namely syntactic and semantic accuracy, in order to give the machine the task humans are much better at, namely weighting facts and drawing conclusions. In other words, it places both the machines and the humans at their maximum disadvantage.

The "subjective" and "assessment" parts of the record must therefore largely remain textual and unstructured[14]. Since it is hard, or even impossible, to assign exactly which subjective complaints belong to each "issue"; the patient's well-being depends on *all* his problems to some degree, this part should be common to all issues. In other words, whichever "issue" I'm focusing on, I should always see the same "subjective" history in full, for that encounter.

[13] I'm using the SOAP terms here, but the same applies to non-SOAP structured records.

[14] Colleague Johan M, and others, beg to differ, thinking that even the patient history should be structured. Only time will tell.

The "assessment" part should be shared the same way, since the assessment *must* refer to all the issues the patient has, else it's not a good assessment.

Taken together, this implies that the complete data for an encounter include *one* "subjective" field aka "history", a set of one or more "issues" which contain the clinical examinations, results, planning, etc, and *one* "assessment" field.

23.7 Finding results

There are several situations where we need to find replies to referrals, x-ray reports, lab reports, and so on. That could happen during a meeting with the patient or when responding to queries from other doctors.

There are three main "angles of attack" when looking for documents, and that is:

- By issue, i.e. everything related to headaches or diabetes. When we try to form an overall picture of the particular issue we're seeing the patient for, this is the most important view. But also, when referring a patient to a specialist for a particular issue, this is also the most fruitful search method.
- By type of document, i.e. x-ray reports, lab reports, replies to referrals, and so on. When considering ordering a test of some kind, it becomes important to see if we have already done that test or examination, or something very similar to it.
- Chronological, that is everything that happened recently, or during a certain time span. This method is more a way of figuring out exactly what was done by us or someone else lately, to see where things are leading. It is often the last choice in searches if the more directed searches don't work.

We refer to results during encounters, since our conclusions are based on these results, among other things. Everything we base our conclusions on, including these results, should be part of the record at that point.

How it is

During the encounter, with or without the patient present, we retrieve a number of results, some of which will be significant for our conclusions and actions. Since it's important to record in the record not only what conclusions we reach, but *why* we reach them, we have to refer to results somehow. In current systems, there's generally no other way to refer to

them except as a textual description. There is then no way for a later reader of the record to with certainly determine what results we used, except by painstakingly going through the list of results and matching them up against our, possibly inexact, textual description. I haven't seen any system where the textual notes allowed a direct embedded link to another document in the records, but if that could be done, it would certainly help some.

How it should be

If we work in an environment with "issue" templates, the template itself will have items where it refers to results. If the diagnosis of an infection involves an estimate of CRP[15], the template has the wherewithal to open up the lab reports, then go look for the most recent few CRP values for you. Similarly, if the template contained a possible referral in an earlier encounter, it will now go look for a possible answer to that referral and present it in the context of the same issue.

When the user then uses the issue template to look up a value and selects the value he chooses to be most significant, that choice is preserved in the issue, making it clear for later readers exactly what result the doctor did use to base his conclusions on.

23.8 Receiving results

We also receive results outside the context of a patient encounter. Lab reports and replies to referrals are brought to our attention when they arrive, regardless of what we're really doing at that point in time. Most doctors set aside some time during each day to go through these new results.

When I see a result from the lab, an x-ray report, or a reply to a referral, the first thing I need to do, even before reading the result, is find out why the tests were ordered, or why the referral was written. I need to locate and assimilate the context in which this examination was required, else I won't be able to understand the implications of the results. Ideally, the result should be presented in a reproduced context, with all the notes, other results, and assessments, that were a part of the thinking when the order was made, regardless of if it was ordered by me or another doctor.

If I don't see that context, it becomes uncertain if I will fully appreciate the meaning of the result and there's a chance I will fail to react to it as I would have reacted if I had gotten the result while being in the state of

[15] C-Reactive Protein, a blood test that gives an indication of infection and inflammation.

mind I was when I ordered it. If I fail in that way, the result will not be optimally useful, and the care of the patient will suffer.

How it is

Most systems present new results in a list of "new results" or "unsigned results", paralleling the old paper based workflow. Back then, we usually got a stack of results dumped on our desks by a secretary, each result attached to the complete medical record of the patient. We were expected to scan the result, do something, then sign off the result in the right lower corner or some other predetermined place on the paper, showing that we did pay attention and from now on, we take the blame if the appropriate action isn't taken.

So now we're in the 21st century, and we *still* are expected to sign off on the result in a very similar fashion. There's no integrated idea of *why* the test or referral was done, or *what action* we could or should take and if that action did indeed result from it all. The system expects one thing, and one thing only: my signing off on having seen it.

The act of signing off doesn't really imply anything. I could act on the result without signing off, or I can sign off without doing anything else with the result. In some systems, simply *viewing* the result sets a flag, in other systems, the user has to click a button or equivalent to "sign off", but there is no relationship between this act and a medically significant action resulting from viewing the information.

Worse, the system presents new results in the context of new results, not in the context I had when ordering the test or referral. This leaves it up to me as a user to go back into the record and figure out why the test was ordered and what to do with the result. Far too often, the test or referral is done by someone else, under assumptions I don't share, with intended actions that were never written down, so I don't know what to do with the result. This is particularly a problem when I'm not private to my predecessors plan of action, and when *my* plan of action wouldn't have included the ordered test or referral. Due to the lack of context and reasoning, the test or referral result will turn out to not only waste my time, but also not result in any useful new plan of action.

How it should be

If a referral or test is ordered from the context of an issue template, that same issue template gets called up and presented together with the result. In this context, the reasoning behind ordering the test is clear and explicit, and the template also contains suggestions on what to do with the result.

What the template does is simply making the plan before, during, and after the test explicit and clear.

If I'm getting the results of a test ordered by a predecessor for reasons I don't agree with, this comes down to not agreeing with the issue template, i.e. the guideline, my predecessor chose, and that is entirely possible and fair. But then at least I know my predecessor followed a plan, which exact plan that was and why, which gives me all the material I need to decide on which plan to follow in the future in a fully informed and considered fashion.

Having the originating issue template pop up when viewing a new result also saves huge amounts of time, since all the tools for prescriptions, new referrals, letters to the patient, etc, are right there, for *that* issue. It also saves brain power, reducing cognitive load, and reducing the risk of missing details and making mistakes.

The result should only be taken off the list of "new" results when it has been used for other actions or documents. There is no point in having a mechanism flag the result as "seen", since having been "seen" means nothing. The user isn't guaranteed, or even likely, to remember having "seen" anything, and even if he did, being remembered isn't something that in itself helps the patient. We should make a direct connection between removing the item from the "new" list and the appearance of a new referral or action in the system. These two events cannot be unlinked from each other.

The list should in fact not be called a list of "new results". It's more usefully referred to as a list of things requiring some kind of action. Note well, that even "ignore" is an explicit action for which the user needs to take responsibility. Doing absolutely nothing is *not* an action, so in that case the item in the list remains in the list until some user takes responsibility for assigning it to an action or an "ignore" category.

If a result or report applies to several different specialities or users, then assigning it to an action by one user, may leave it in the unassigned list (the incoming "new" list, if you will) for any other users considered relevant as receivers of the result or report. This avoids having the result disappear from under a user even in the absence of action.

23.9 Reporting

Reporting encompasses mandatory reporting of communicable diseases, but also statistical reporting, and care burden related reporting. Reporting related to national registries are treated separately, see Reporting to national registries (section 23.10).

To do the reporting right, all the elements needed in the report must be available. These elements often aren't part of the clinical examination and history taking of any particular encounter. To avoid poor reporting or extra work, either all the data must be captured when it's available and used for later reporting, or the reporting should be done during the encounter with the patient.[16]

How it is

Reports and attestations of all kinds are usually implemented in current EHR systems as a collection of forms with pre formatted fields. Some of those fields are filled automatically, such as patient demographic data, current date and time, doctor's name, and so on. Most other fields are left empty, such as "history", "clinical signs", "conclusions", "recommendations", etc. These fields are often of a character and intention that is dependent on the use of the document.

The same problems I referred to while discussing prescriptions (section 23.5) appear when creating these documents as well. If I'm in the process of handling a particular problem for a patient, only a very small subset of documents are relevant. If I'm treating an infection, reports for communicable diseases may be relevant, but a certificate for a driver's license is likely not. In current EHR systems, I will still be presented with all relevant *and* irrelevant document forms to choose between, greatly increasing the time and irritation in finding the right one, while at the same time increasing the risk that I'll choose the wrong form. Which, of course, will come back and bite me later.

Once we get past the onerous locating of the right form, we have to fill it in. This usually consists of going back and forth a great number of times to look up information in the records, then copying and pasting it into the form, occasionally editing the text to look better, or to fit the form field.[17]

How it should be

Again, if the user is working in an "issue" template for the current problem or disease, any document forms that can be relevant for this issue will be listed there, and can be activated from there with a click. This limits the set

[16] Sometimes the need to report only becomes obvious once results come in, and in that case, it's likely some elements needed for the reporting were not captured, and we have to contact the patient again for that.

[17] As I'm writing this, I feel my rage boil up again at that one system that for years didn't allow switching back to the record, then didn't allow copy and paste, and *still* even with copy and paste can't handle an excess of text for a form field, simply refusing to paste in that case. Oh, and it won't allow copying less that a full field from the notes either. This is the way you turn plain bad software into an epic fail.

of possible forms to those that actually make sense for the issue. Nothing hinders the user from going to the full library of forms if he needs something exceptional, but that would rarely be needed.

Since the form is linked to the issue, it is easy to link the contents of most form fields to existing items in the issue template, eliminating most of the copy and paste work. Only very few fields will ever need to be entered fully by hand.

Having the form created from the issue also means that this relationship is preserved. It will be trivial to see in the future that a certain document or report has been created for this issue, and conversely, what issue was the reason for the issuing of a particular report.

23.10 Reporting to national registries

National registries are maintained for selected diseases, such as heart failure, diabetes, hip replacement surgery, and more. These registries can be useful for a number of statistical purposes, and in selected cases may even contribute to scientific knowledge.

Often reporting is done after the patient encounter by reading the medical record and extracting the reportable data from it. The problem that often occurs is that the right data was not collected during the patient encounter, necessitating contacting the patient again, or submitting incomplete report forms to the registry.

If the reporting can be done in the presence of the patient, or forms (paper or electronic) are provided for the doctor to use during the encounter, the frequency of missing data can be reduced to practically nil.

How it is

In current systems, there is either no help in creating reports to national registries, or forms are provided in the notes with all the data fields needed for these reports.

Having the fields in the EHR clearly makes it easier to create complete records with all the required data during the encounter. Better systems also eliminate duplication of entry between the normal clinical examination and the reporting form.

These forms are then checked and completed by a nurse or doctor, then sent to the national registry in binary form (hopefully), and anonymized at arrival. In some systems, the patient identification is transformed using some form of hashing function before being sent to the registry.

How it should be

If all required fields for the national registry is part of the clinical issue template, the creation of a report is a simple question of extraction of those data values, and subsequent packaging into a reporting data packet.

Before transmission, the data should be fully anonymized and provided with a one-time-only patient identifier, that can only be resolved to the right patient by a designated third party. See my section on anonymizing patient data (section 34.1). For a technical solution, see the appendix on registry anononymizing, page 237.

23.11 The real requirements

With the lessons of the previous chapters in mind, how do we formulate the *real* requirements for an EHR system, that includes the features we need, while not needlessly limiting the solution space? I'll touch on the areas I think are most important.

I'll formulate these requirements as "awareness" key points, underlining the importance of keeping the doctor or nurse informed of the right things, not *how* that information is brought across. It does not matter in the least if it is presented on paper, on a screen, by needle pricks on the back of the hands, or by telepathy. Those are all possible (or impossible) implementations fulfilling these requirements, and it's up to the fantasy and ability of the architects and designers to choose the method.

In the same fashion, input to the EHR system can take any conceivable form, such as typing, speech, gestures, telepathy[18], interpretive dance, or music, but the main point is that it is up to the architects and designers to invent suitable methods. The key requirement is that there *is* an input, not *how* it's done.

Awareness of issues

When seeing the patient, the doctor or nurse must be made aware of all issues the patient has or has had that can have any relevance to the current encounter. The doctor or nurse must also be made aware of the issue that is the subject of the current encounter.

An "issue" consists of a short description of a medical problem or fact that forms the subject for diagnosis or treatment. It should be distinct enough to allow identification in literature, as indication or contraindication for medication, and as the basis for public health reporting or statistics. It

[18] You wish.

should be of a form that is, or could be[19], available in standard coding systems such as ICD–10 or SNOMED CT.

As the user is made aware of the issues, the user should, for each of those issues, be made aware of all the diagnostic and therapeutic steps that have been taken in relation to that issue *and* the reasoning behind it. That "reasoning" includes at least the planning and what scientific base that planning has, including both diagnostic and therapeutic planning.

Any "summary of care" documents belonging to any issue should also be clearly presented. There should be a way of finding more details than the summary of care presents, but those details could be provided through other means.

Any documents forming the basis for conclusions in summary of care documents, or in responses to referrals, should be included directly or indirectly in these summaries or responses. See the appendix where I discuss referring to sources in the document tree design, on page 251.

Awareness of patient history

The doctor should quickly and painlessly be made aware of the patient's history in general, i.e. those aspects not tied into a single healthcare issue, but more related to the whole patient. Things like general well-being, ability to lead a normal, or at least satisfactory, life, and the major obstacles to that, including social and financial. This history should not be fragmented and contradictory, but be presented as a consistent whole, where not only the different aspects, but also the evolution over time is clearly shown.

Yes, that was an uplifting mission statement for our systems, but that is actually what we need to accomplish. Exactly *how* this is done must be left up to the designers of the systems.

Awareness of planning

The doctor needs to be made aware of the plans used in diagnosis and treatments, what these plans consist of, the sources they are derived from, and how far along in these plans the diagnosis and treatment have come.

These plans must be explicit and detailed enough so they can be compared with other plans and "current best practice", both by the doctor and the patient. The sources must also be explicit enough so that it can be verified that these plans are not invalidated by any sources having been retracted or superseded by more recent science.

[19] If it could be, but isn't available in standard coding lists, one should take possible mechanisms into account to add the issue to these coding systems.

It should also be clear exactly why these particular plans were selected for this patient, if it was due to the location, the patient's own characteristics, or a preference by patient or doctor.

Awareness of outcomes

It should be clear from the overview what the status of issues is. Are they active, and if so, who is responsible for this issue (doctor, provider, institution)? When was it last managed, and is there reason to think that it needs more attention? Has it been forgotten about and left without action for too long? Or is the issue resolved, and if so how (briefly)? Or are there outstanding results or responses related to this issue that need handling?

Ensure action

When results or replies to referrals become available, the user should be made aware of these. The system should keep showing these results or replies as "new" until effective action has been taken, and these results have become an integrated part of further decision making. The simple viewing of the result should not be regarded as effective action.

Creation of issues

The user should be able to define "issues", that is healthcare related considerations about the patient. These could be diseases, problems to solve, incidences, states such as pregnancy, or actions such as vaccinations.

It should always be possible to record at least a basic history and clinical examination without first defining an issue. Early in an encounter, it can be difficult to determine which issues are relevant.

When defining an issue, the user should be able to locate the right issue from a list of issue templates in the system. It should be searchable alphabetically, but also on keywords that the user can enter, and on keywords extracted from the history already entered or from entire note entries from older records in legacy systems, if available. The idea here is to reduce the total list of available encoded issues to a manageable smaller collection, excluding issues that clearly has no bearing on the current case.

The issue template should also be findable by origin, such as the local institution, local area, regional, and national level. If available, it should also be searchable on an international scale.

When the template is located, it should be useable as is, or the user should be given the opportunity to add items to the template, or to mark items as not appropriate for this patient, or even change items. The idea

is that the standard template should be a starting point, not a mandatory ruleset.

When creating an issue, it can come with a template for clinical examination, referrals and orders relevant to the issue, diagnostic codes that can be relevant, reporting templates, and more.

Deletion of issues

The user should always be able to remove issues, since further workup may show that the issue wasn't correct. A tentative diagnosis may not be confirmed, or it may be refined and narrowed to a more specific issue.

When removing an issue, items consisting of clinical findings that were already entered by the user should remain in the record. If these items do not appear in any other issue, they should be listed in a separate summary for the encounter, such that all entered clinical findings are always visible somewhere, in at least one context.

Removed issues should be listed in a separate audit log, such that the sequence of adding in issues and then deleting or replacing them, should be easily traceable if needed.

Creation of issue templates

Issue templates should be created by the users themselves, and stored in such a way that they can be easily reusable by the creator and by others. When creating an issue template, the user can either start from scratch, or start from an existing issue template, which is then modified to suit the user.

An issue template consists of an order series of *issue items*. Each item consists of a *prompt* and an optional list of *preset values*. For instance, if given the issue item shown in figure 23.2, the "Limb ataxia" text on the left is what I call the *prompt*, while the list to the right contains three alternative *preset values*[20].

When creating these items in a template, the user can choose the prompt from a list of possible prompts according to language. These "prompts" are often what are called "clinical findings" in other places. They're elements in SNOMED CT, for instance, so a mechanism should be made available to choose from a code table.

The *preselected values* should also be selected from preexisting and relevant code tables, if available.

[20]Note carefully that this description and figure only form an illustration of one particular implementation, but is not a requirement defining how it should look. The particular method of implementing issue items should be left up to the creativity of the designer.

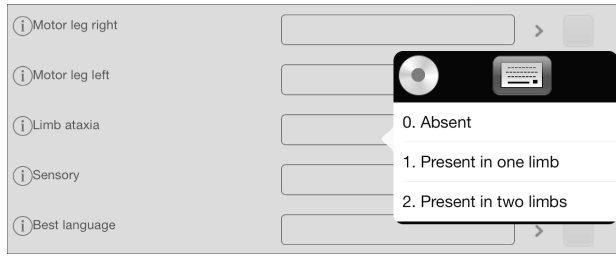


Figure 23.2: Issue Template Item

Both the prompt and the preselected values can be present as uncoded text, as a code from a reference code table, or with more than one referenced code.

If entered as text, the text can be entered in several languages, such that automatic translation of these templates can occur, both for the prompts and the preselected values.

Since the user is always free to enter free text or a dictation into fields, there is no provision for mandatory text entries. The user is also always free to leave fields empty when using a template[21].

Figure 23.3: Issue Template Subtitle

All items should have an optional information element, that should also be multilingual. That element could contain text, images, or be a link to a local or remote site. Care should be taken not to disable the functioning of the client system if a link is not resolvable[22]. You can see the presence of such links as a circled "i" in the left margin of figures 23.2 and 23.3.

[21] Resist the temptation to allow mandatory fields, just because management wants measurements. It will only alienate the users and cause unreliable data to be entered.

[22] Not all networks work all the time, nor do all websites stay up forever. Handle it.

The user should also be able to create subtitles for better visual effect, see figure 23.3, top row.

Recording of history

The "history" element is by its nature free text. There is no useful way of structuring this according to some predetermined syntax. Its main function is to relate what the patient experiences with as little transformation or interpretation as possible.

Since this element relates the experience of the patient as a whole, it cannot usefully be assigned to any particular subset of issues the patient has, so it belongs to no single issue in the record. It can be a part of any or all issue templates, but the content of the element will always be the same across all issues for any particular encounter.

All this implies that the history part of the record cannot be made available in multiple language versions, unless there's a translation tool provided with the system.

Recording of clinical examinations

Clinical examinations are not in essence different from other elements in an issue template. Each clinical item consists of a prompt and optionally a set of preselected values, as described in the section on creating the issue templates (section 23.11).

When designing the template, one of the preselected values can be indicated as a "default" value, and the system should have a convenient, easy to remember, and fast shortcut or gesture that activates the default choice. Going through a clinical examination, or any other kind of questionnaire, that consists entirely of default entries, should be optimized for speed. In our *iotaMed* implementation of issues, the user double-taps an item to select the default value.

Keep in mind that the absolute majority of clinical examination items, even in a very ill patient, will be normal, i.e. correspond to the default value, so the system should optimize for this path.

Don't lead me up the garden path

Some legacy EHR systems have warnings for potential errors. The most commonly implemented is the pharmacological interaction warning, which I'll take as an example of everything that is wrong with how these systems are built.

What happens is that you are first presented with a list of *all* medication products you can prescribe, then you're allowed to select one, and only

after that will the system come back and tell you it's a bad choice[23]. Now, this just serves to make us hate the system.

What the system *should* do, of course, is only present you with the products that are relevant to the issue you are in the midst of working through. And even *if* it would present pharmacological products that have a contra-indication or interaction warning attached to them for this particular case and point in time, that warning should result in a flag that is visible at the point in the issue where you would prescribe that product. In other words, don't waste my time selecting a product the system *already knows* has a warning attached. That warning should be visible *before* selecting the product.

Products that belong to the normal arsenal for the treatment of this issue, but that should not be prescribed due to contra-indications or interactions, should still be displayed, albeit with a flag of some kind. If you hide these products, you'll only confuse the user, since the disappearance of a well-known therapeutic product from the issue template could be misinterpreted by the user to mean that the template is defective, or the product withdrawn. Also, the interaction or contra-indication warning is just that, a warning, and the user may need to prescribe that product anyway, after considering the alternatives.

The user should always have the ability to choose any product from the total list of existing products, but that choice may reside one or two levels deeper in the interaction hierarchy, since it should rarely be needed if the issue template is well designed.

Confidentiality

The system must allow setting access limits on issues. This is what I call "medical confidentiality" in the section on that (section 34.1) earlier.

This confidentiality flag should limit access in several levels to groups, or roles, of users. The granularity should not be too detailed, since that makes the system hard to manage, so a bare minimum could include the following levels:

- Accessible only to the creating organization (the department handling the problem).
- Accessible to the above, plus designated individual doctors/nurses, or designated other organizations (care centers, departments).
- Accessible to all authenticated staff.

[23] Adding insult to injury: many, if not most, of those warnings are so wrong they're just a waste of time.

Since the access restrictions are set on the issue, not the department, some issues can be confidential, while other issues are not marked as such, even though both kinds can originate in the same department.

The confidentiality setting also includes all medication, all referrals and lab reports that are originated from the issue in question.

Since the presence of an issue, or any result covered by the confidentiality setting of the issue, can form a warning or contra-indication when another doctor or nurse prescribes medication or orders diagnostic tests or treatments, the system must still be able to warn for that. If the initiating doctor or nurse are excluded from viewing the issue, one of the following actions can be prescribed by policy to occur:

- The originator will be warned about the contra-indication, and told what it consists of.
- The originator will be warned, but not told what it consists of.
- The originating organization for the hidden issue will be told there is an attempt to prescribe an action that could be dangerous, and this organization will have to resolve the issue somehow.

Other combinations could be possible, such as involving the patient, or involving only the actor that created the confidentiality flag in the first place.

Chapter 24

Encapsulation

If we forget about healthcare for a minute, and look at how information is handled in other parts of science and society, we can see that as the information and knowledge volume increases, compartmentalization and delegation compensates for complexity and allows us to keep evolving. As an example that should be easy to visualize, let's take programming. Without going into the entire history of computing, we can claim with confidence that one of the most important steps in the evolution to highly complex systems was the invention of object oriented programming. Object oriented programming (OOP) is based on the following ideas:

- The details of an implementation should remain invisible to everyone using an object in a larger context, so much so that the internal coding, the implementation, can change as long as the use of it by other objects does not change.
- The interface to the object should be as minimal as possible and contain nothing that depends on the exact inner workings of the object.
- At each level of abstraction, the programmer composes objects and creates a new object with a higher level of abstraction.

This way, OOP leads to ever higher levels of abstraction, each level being free from knowledge of details of objects at lower levels of abstraction.

Correctly done, OOP removes from view all the internal complexity you don't need at any particular level of complexity. This staged reduction of detail, opaque encapsulation, is what turns a potentially exponential growth of detail into a linear process that can be handled by human minds.

In medicine, the same process applies. The patient talks to the general practitioner (GP) using a high level API: "I'm sick. I think it's my liver. It runs in the family."

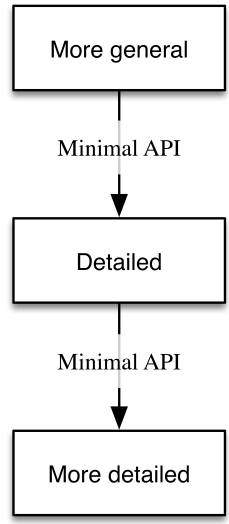


Figure 24.1: Encapsulation flow

The GP reasons to himself in much more detail: *"The liver. Right... What he probably means is that he's prone to nausea and stomach aches, but that's probably because his whole family is living off hamburgers and get into drunken fist fights over the TV remote. Anyhow, I'll check his transaminases, not forgetting the gamma-GT."* But what he says to the patient after poking his abdomen for a bit is something like: *"Hm... your liver is maybe a little tender, we'll run some tests."*

If the GP had gone to a continued professional education (CPE) class and learned that there is now a virus causing a deadly disease involving symptoms of nausea, fights, and remotes, and that it was brought to earth by the moon landing crew[1], he would still have responded the same way: *"We'll run some tests"*. In other words, the change in the GP's internal implementation of how to do medicine in a family practice in the space age does not lead to a change in his information interface towards the patient. The GP is fully encapsulated and OOP compliant.

Let's push this example a bit and assume something is terribly wrong with the lab tests and the GP refers the patient to a gastroenterologist with the general question: *"What's up with this liver? Maybe a biopsy would be a good idea?"*

[1] Don't worry. I'm lying.

Now, let's further assume that the gastroenterologist agrees[2], goes on to make an appointment for a biopsy, chooses the most suitable ultrasound transducer, the right needle gauge and length, etc, etc, and does the biopsy. The gastroenterologist, in turn, sends off the specimens to a pathologist for microscopic, and possibly histochemical, analysis.

The pathologist also does his thing coloring, embedding, slicing and dicing, etc, etc, and all this results in an answer from the pathologist to the gastroenterologist detailing the microscopic findings, and elaborating on the type of tissue damage seen. The terms "cirrhosis" and "steatosis" may have been used in this report. A few other minor types of cell damage or proliferation is also noted.

The gastroenterologist looks at this report, decides from the clinical picture and the pathology report that the cirrhosis is significant, the steatosis less so, since it's very common in the general population, and that the other findings of minor cell damage are coincidental and irrelevant for the current major complaints. So the gastroenterologist reports the most significant findings only to the GP, and just takes notes of the less significant findings in his own notes, just in case they will turn out to be relevant later.

So, in summary, the gastroenterologist reports to the GP something like this: *"The biopsy showed a moderate degree of cirrhosis with some steatosis".*

At this point, the GP is supposed to understand *"cirrhosis"* and *"steatosis"* and more or less what to do about it (cut out the alcohol and the hamburgers, the fist fights are no problem). But the GP does not need to know how to do a liver biopsy or how to prepare the samples for microscopy or even how cirrhosis looks in a microscope. Even if the specialist buys new equipment and then does his biopsies in a different and better way, this makes no difference in the interface between the GP and the specialist. In this example, the gastroenterologist is fully encapsulated versus the GP.

> *This encapsulation allows every layer of abstraction to evolve independently. The GP can change and improve his methods without the patient noticing*[3]*. The specialist can change and improve his methods without changing his interface towards the GP. This is the only way to allow medicine to evolve, going from the "super GP" who knew all of medicine in the middle ages*[4] *to the super specialists of today.*

[2] This does happen.

[3] Except as better, quicker, and perhaps even more gratifying encounters with primary care.

[4] Which wasn't much.

It's relatively easy to see that the same process of layered levels of abstraction of knowledge applies in all intellectual human endeavors, not only programming and medicine.

And here comes the moral of this story:

> *To allow medicine to work efficiently, we must mirror the same levels of abstraction, encapsulation, and separation of concerns in the EHR as the EHR becomes our primary tool. Only high-level information should be shared by default, not the details. If we keep flattening the EHR as is generally done today, with access to every detail at every level, we're moving medicine back into the middle ages instead of forwards into the 21st century.*

And, more bluntly:

> *Large, unified EHR systems are a really bad idea. A much better idea is loosely coupled specialist systems, each with a narrow interface, mirroring object oriented systems and allowing full knowledge encapsulation.*

Exercise for the student: how does this destroy the idea of allowing the patient access to the EHR as it is implemented today?

Chapter 25

How active should the software be?

When the software contains information about diagnostic criteria and at the same time has the clinical data pertaining to the patient, it is tempting to assume that we should let the software draw conclusions from the clinical data and establish diagnoses. Making decisions is what computers are for, after all. But designing for that would be a mistake.

25.1 The keyhole effect

If we let the software make diagnostic decisions and pose questions according to past answers, it will lead us through a series of questions and answers that according to its programming. It proceeds along a path through a flowchart. This hides the overview of the process from the user, giving the user just a "keyhole view" into the exact questions and parameters that the software deems interesting at any particular point of the decision process.

As a user, I'm inclined to game a system like this, simply to be allowed to view the different branches of the decisions tree that are hidden depending on particular data input. The whole thing quickly degenerates into a charade of false inputs just to make the desired information come up on the screen. Putting in false information into the record for this reason, or for *any* reason, is a really bad idea that is bound to come back and bite you later.

25.2 The indiscriminate criteria effect

When the software chooses clinical data to match to criteria, this is often done quite mindlessly, leading to wrong conclusions. For instance, the

criteria for the diagnosis of diabetes is, among other things, "two consecutive capillary glycemia values of 7.1 mmol/L or higher". If we let the software make that diagnosis based on the series of glycemia values in the records, it will make that diagnosis in many cases where a doctor would not, and vice versa. Many of these values may be non-representative, the result of other influences that the doctor knows about, but which the software doesn't. Also, if the software does *not* make the diagnosis, but the doctor does, it will probably force the doctor to falsify clinical data to make the software behave as he wishes it to behave.

If the doctor is made to fiddle with data to make the software draw the right conclusions, the set of clinical data becomes suspect. The right role of the software is to present clinical data and criteria together in an easily digestible format, aiding the doctor as he draws conclusions and makes decisions.

25.3 The disempowerment effect

If the EHR system is enabled to make decisions that used to be taken by doctors, healthcare providers may see this as a way to reduce the dependence on doctors, thereby increasing the capacity for healthcare, or reducing the costs for doctors, or both. There is nothing wrong with these goals, but there is a significant probability that doctors will see this as disempowerment and refuse to delegate that power to the IT system.

Since getting the cooperation of doctors is crucial to any successful automation project, you should carefully consider if it is worth it to pursue a transition plan that involves taking power away from doctors by force. It is probably more prudent to have that transition occur further in the future, as it will undoubtedly sooner or later be the case, and have that transition be initiated by the doctors themselves.

25.4 Nurse vs doctor domain expert

The user that is taken as a reference when designing software is called a "domain expert". I argue elsewhere (chapter 20), that this user should also be a requirements engineer or system architect, but that is not the point I'm belaboring here.

The leading user influence in EHR development is usually either a nurse or a doctor, and it seems that the difference between the outlook of these two professional groups is severely underestimated.

Nurses usually work in a process oriented workflow: start from the top and work your way to the bottom of the list. Typical examples is preparation for operations, postoperative care, post anesthesia checks, etc.

Doctors seldom work in a directed workflow, but tend to work with a list of things that should be considered or done, and where the order, or even completeness, is of secondary importance.

A software system designed by nurses will have a fundamentally different workflow from a system designed by doctors. There is nothing wrong with this, unless you let nurses design systems for doctors or vice versa.

Chapter 26

Document tree

In the section on receiving results (section 23.8), I described how each received result must include references to all other documents and results its conclusions are built upon. In the document structure we have in current systems and that I described on page 79, that is quite impossible to achieve. The relationships simply aren't in place to do that.

What we need instead is a document structure in the application that mirrors how decisions and documents in clinical practice depend on each other. As an example, when I write a referral to a specialist for a patient with some as yet undefined problem of the liver, I'll probably send along some lab results and the protocol from an ultrasound examination of the liver. This could diagrammatically look like figure 26.1

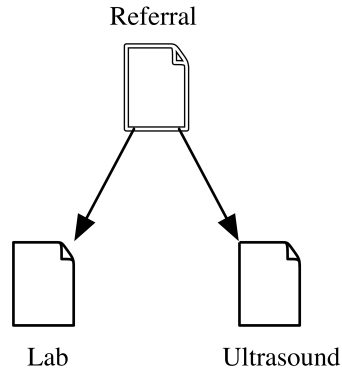


Figure 26.1: Referral with two included documents

In this figure, the top-most element has a double border to designate that it is a "root", in other words, an element that is not owned or included by any other element. The significance of this will later become clear.

In my example, the specialist performs a liver biopsy and sends it off, then later gets a report from the pathologist about the tissue sample, then finally uses that tissue report and the ultrasound protocol I sent him, together with his best judgement, and writes me a reply to my referral. His reply includes the pathology report and the ultrasound protocol I sent him. His reply will also refer to and include my initial request. So what we'll see in the record system after receiving the report will look like figure 26.2. Also note that the report from the specialist is now a "root" element with a double border. The request I sent to the specialist has lost its "root" double border since it is now referred to by another element, the report.

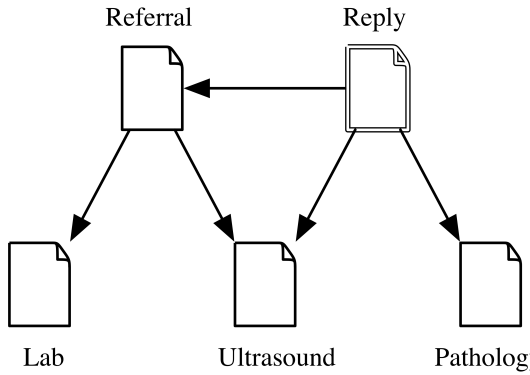


Figure 26.2: My system after receiving a reply to my referral

If the system I'm using had a list of "root" elements, that list would initially hold just one entry, the request in figure 26.1. Once I got a report back from the specialist, that entry would disappear, since it's not a root any longer, and be replaced by an entry for the report back from the specialist, as in figure 26.2.

So, what does this mean in clinical terms? Each root is an independent problem, an issue, something we should keep an eye on or react to. After writing a referral, the referral becomes an item to watch, to keep an eye on, until it results in a report back. As soon as the report comes in, the referral can be removed (which it automatically is, since it becomes referred to by

the report) and does not need any attention anymore, but the report itself now becomes an item of interest. That report *remains* an item of interest, a "root" element, an item in the list, until some other item is created that makes use of it, and it becomes part of a larger, higher level reasoning.

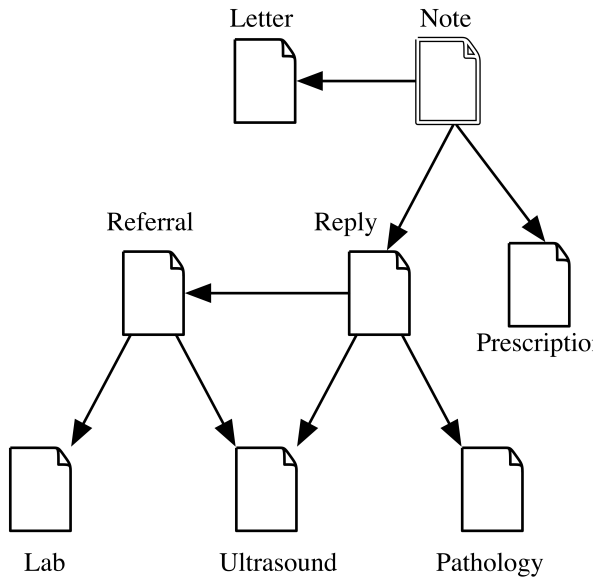


Figure 26.3: The document tree after writing the note

If I read the report from the specialist and make a note of that in the records, write a letter to the patient, and create a prescription, while referring to all three in my note, I will have replaced the root, that is the element to keep an eye on, with that note. The new relationship will look like in figure 26.3. In this tree, the note itself becomes an "item of interest". Clinically, this makes sense, since we need to act on it, make it part of something else, classify it as part of a healthcare issue. Maybe that issue could be "liver problem". If we add that high level root to the records, it will look like figure 26.4.

This high level healthcare issue could also be diabetes, hypertension, or schizophrenia, for instance. It could also be a symptom which hasn't been clearly assigned to an issue yet, such as "headaches" or "exhaustion". It could also be an incoming request that hasn't been seen and handled yet.

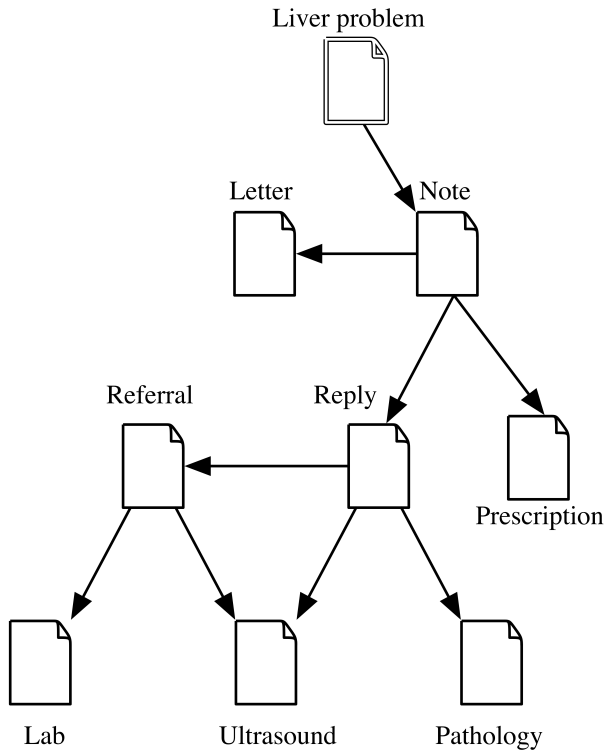


Figure 26.4: After creating the issue "Liver problem" at the top level

The list of root documents forms an *excellent* overview of the patient's current issues and any elements that are not yet referred to by any other elements. New incoming results will therefore automatically show up in this list and stay there until someone acts on them. As results are acted on, and therefore linked into other actions as subdocuments, they disappear from this list. The only way to remove an element from the list of things "to keep an eye on" is to act on it in a way that makes that element part of a whole. The very action on the element removes it from the list. The list fills many of the functions we see in the list of "unsigned" items in current systems, but in a much more sensible way.

Another way of looking at the inclusion of documents into other documents, is that they are interdependent. The higher level document *depends* on the lower level documents. If any documents that contain judgement calls or measurement values turn out to be incorrect, then any other docu-

ments that refer to them also become suspect. This is inherent in medical reasoning and should be reflected in the document architecture of the EHR system, as it is in this design.

If we start from the top, instead, then we first read my note with my conclusions about the liver problems, and from there we can find and view underlying documents, the one from the specialist, and in turn the pathology report. Clearly, we can reach *all* details that were used in any conclusions this way. Any conclusions that are *not* adequately based on other findings will stand out as a sore thumb.

Interestingly, we can *only* reach the details that have a direct or indirect relationship to the top level issue we start with, which eliminates a lot of irrelevant information from the context of a particular issue or result. The details that we don't see as part of an issue tree will be shown as part of another issue tree, or if they are entirely free-standing, they will be shown as roots in the attention list.

Elements can have several parents, i.e. be part of several trees at the same time. This is logical, since the same lab result or specialist report can have bearing on more than one issue. Everything that shared element depends on will automatically also become shared between both issues.

26.1 The attention list

I've mentioned the "attention list" several times now, but it's necessary to expand on what this means. To do that, I'll go through the same example I used in the preceding to illustrate how the document tree is built, but this time accompanied by a description on how the "attention list" evolves.

The "attention list" fills the function of overview over the patient's issues, while at the same time filling the function of a list of "unsigned items" as it is implemented in current systems. In fact, it turns out not to be a definable difference between "issues" (or "diseases", or "problems") on the one hand, and "unsigned" incoming results on the other. Both concepts are primary elements of attention and have many similarities from an information conceptual standpoint.

In figure 26.5 we have just created the referral and included two documents in it, a lab result and an ultrasound report. The referral itself is a top level element, a root of the tree, and will therefore also appear in the attention list at the top left. When the doctor opens the record, her attention will be drawn to this referral, indicating that it hasn't been replied to, or made part of another attention item in any way. It's freestanding and calling for attention.

In figure 26.6 the issue "Liver problem" has been created by the doctor, and the referral she just wrote has been made a part of that issue. Since

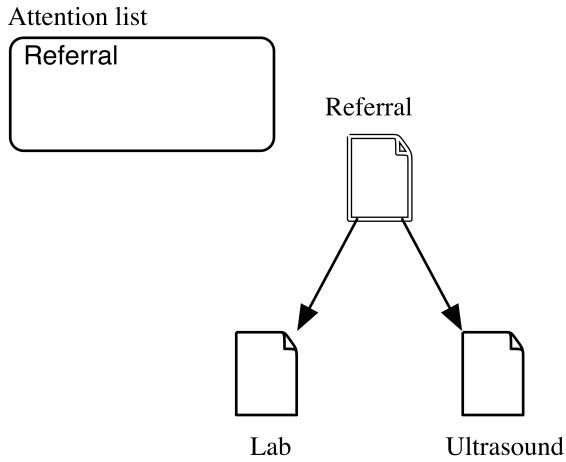


Figure 26.5: Having only written the referral

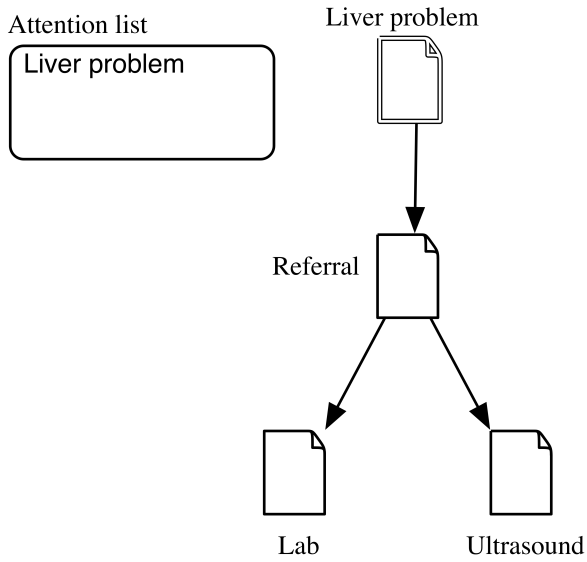


Figure 26.6: The referral has been made part of an issue

the root of the tree is now and issue element called “Liver problem”, that is the only attention item shown in the list in the upper left.

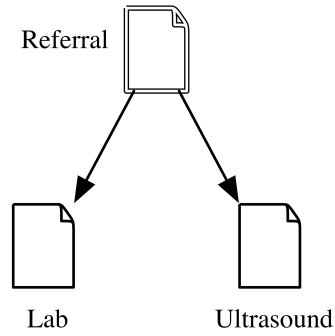


Figure 26.7: The document as received by the referee

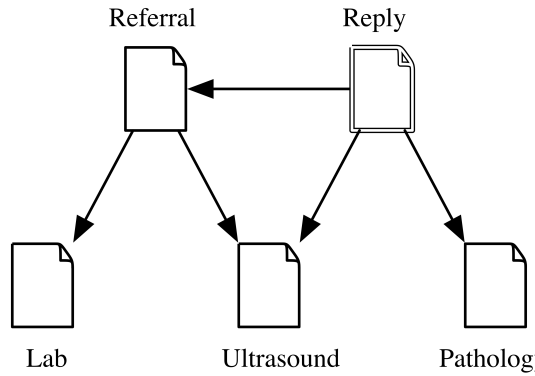


Figure 26.8: The document as returned by the referee

The specialist receiving the referral, the “referee”[1], receives one document with two subdocuments as shown in figure 26.7. The referee then

[1]“Referee” is ambiguous, I know. It could mean the patient being referred, or the doctor receiving the referral, but in this text I will use it to mean the latter. I have no other word for that doctor, while the patient can always be called a “patient” instead. So that’s what I’ll do.

creates a reply document which contains links to the referral document, the pathology report, and the ultrasound report that was sent in by the referring doctor. The document set that the referee finalizes and send back looks like in the figure 26.8.

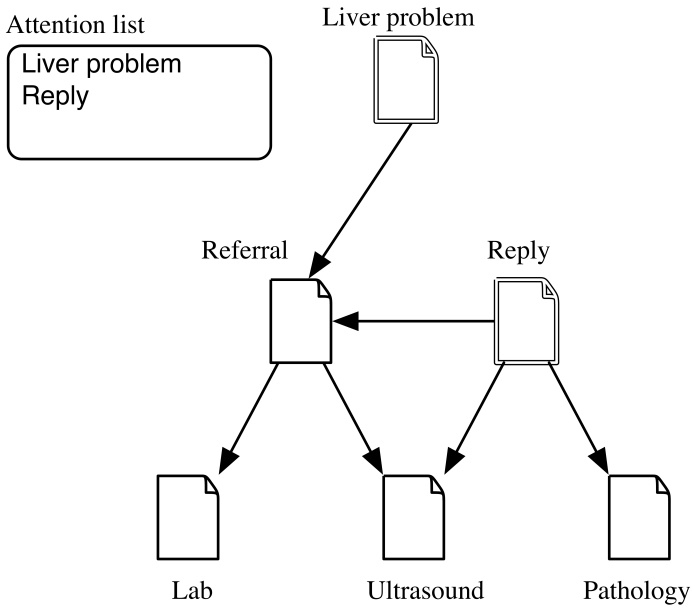


Figure 26.9: The reply has arrived

When the document is received by the referring doctor, it is linked into her system as shown in figure 26.9. Both the referral and the ultrasound report were already present in the receiving system, and are not duplicated. The reply is simply linked to those preexisting documents. The pathology report and the reply itself are added to the document tree. Nothing in the existing tree refers to the reply element, so it becomes a second root of the tree (the first one is "Liver problem"), and is added to the attention list. The fact that the reply refers to elements that are already a part of the tree (the referral and the ultrasound report) does not in itself make the reply element part of the tree, allowing us to have it shown in the attention list.

When the receiving doctor looks at the attention list, she can select the reply from that list, write up a note with her conclusions, base it on the reply, and then write a letter to the patient, and a prescription for a suitable

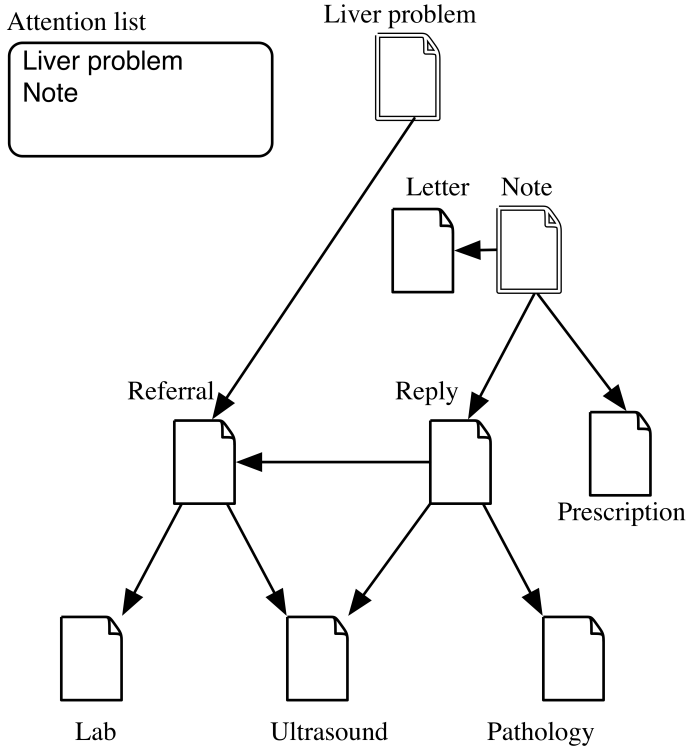


Figure 26.10: The doctor creates a note, a letter, and a prescription

medication. The result will look as in figure 26.10. The very act of basing her note on the reply to the referral makes the reply a child node and removes it from the attention list automatically. The note itself becomes a new root and will show up in the attention list.

The note is in the attention list since it hasn't been yet made a part of a greater whole; it hasn't been properly put into a context. The doctor now selects the note, then links it to the proper issue, in this case the issue "Liver problem", and so makes it a child of "Liver issue" and automatically removes it from the attention list. The result is an attention list containing a single item, the issue "Liver problem", with no other outstanding items that need attention. In a real implementation, this step would almost certainly be part of the actual writing of the note in the first place.

One question arises, namely how to view the letter to the patient and the prescription. Is the letter based on the note, or is the note based on

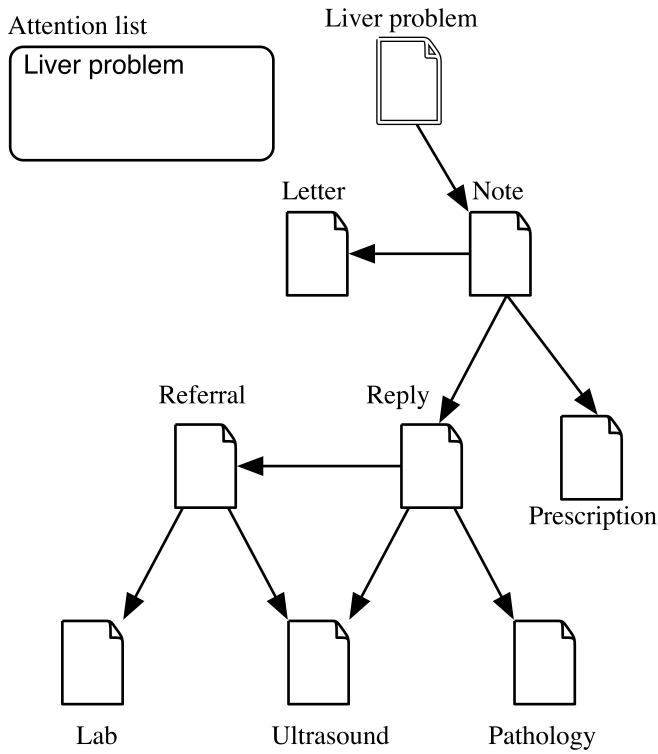


Figure 26.11: After the note is linked to an issue

the letter? The same question arises when we think of prescriptions. It is clear that if the letter has no relationships to anything, it should be in the attention list so that the user is prompted to place it into the right context. The same goes for prescriptions. But if there is a link, which should be child and which should be a parent?

It turns out that elements such as letters or prescriptions are a bit peculiar, they're not below or above other elements such as notes, but more to the side of them. Since it would be clinically absurd to put the letter as such as an attention item, or the prescription as an attention item, I've chosen to always make them children of any relationships they are in.

26.2 Encryption

We can add an encryption twist to this tree. Assume that each included document at any level consists of a reference to the included document *and* a decryption key so that the referred document can be read. This makes it trivial to read any included documents if you have access to the document that includes them, since that is where the decryption key resides. This also makes it impossible to read a document if you haven't retrieved it by way of another document that includes it. The only way to enter the tree is by way of a top level document which can be found in the list described in the previous section. From there, you can descend the tree (yes, it's upside down with the root at the top), accessing underlying documents one level at a time.

It turns out that this is a very desirable property. If a document has been used as basis for another document, that base document will always remain accessible to that derived document, but not to any readers who have no access to a derived document. This sounds more complicated than it is. In more clinical terms, you can say that if you have access to an issue, such as diabetes, you will automatically have access to everything that is relevant to that issue. If you have access to another issue, such as schizophrenia, but not diabetes, you will not have access to any documents that are part of the diabetes tree, *unless* those documents are *also* part of the schizophrenia tree. You will not automatically know that a particular document is part of a tree you have no access to, but that's how it should be. Parents know about their children, but the children do not know about their parents.

The document tree design also neatly implements the abstraction and encapsulation I discussed in the section on encapsulation (chapter 24) earlier. It turns out that this abstraction and division into levels that I described in some depth in that section happens continuously with every new referral or response. The document tree design very closely mirrors how we think about referrals as doctors, which is very different from how it is implemented in current systems.

In a paper based medical practice, I would write a referral or response and physically include copies of documents I refer to, such as lab reports and x-ray protocols. Implicitly, if you have access to the referral (by opening the envelope), you have access to the copies of documents I included in the same envelope. Since the receiver will base his or her conclusions in part on those included documents, the receiver needs permanent access to those documents. In other words, once you've sent them, you shouldn't be able to take them back.

In current computerized medical records systems with electronic transmission of referrals and responses, this generally turns out to be impossible.

There's no provision for attachments in that sense. The solution for the problem seems to be to create huge stovepipe systems, such that the receiver of a referral or result can go scrummage for himself in the record system of the sender for those documents that couldn't be attached in the first place. Worse, this seems to be the *major* reason for large unified systems.

> *In other words, one basic design flaw is incompletely, but expensively, compensated for by another, even bigger, mistake.*

This poorly conceived arrangement leads to absurdities. In order to give the receiver access to material I'm referring to, I have to allow the receiver *full access* to a large part of the sender's system. If I ever revoke that access, the access to the referred documents is also revoked, removing any possibility of later, on the receiver's side, verifying that the conclusions were correctly based on included documentation.

I'll describe the implementation in detail in a later chapter, so the interested reader can see how the whole thing really works.

Chapter 27

The size of the system

One of the major discussions going on about IT systems in healthcare is how large the systems should be. Should we have a collection of specialized systems interacting to form a whole, or should we just have a single huge system doing it all?

When you have problems with getting data from one application to the other, you can be excused for assuming that having all different applications come from one vendor, or be part of a suite of integrated applications, would solve your problems. But nothing is that simple. Since the repeated failures of EHR systems due to this misconception seems not to lead to a realization of the source of the problem, I'll take examples from other system families to illustrate the bad thinking behind these system suites.

The first, and really major, example is the Internet. It's the ultimate example of letting a huge number of vendors deliver an even larger number of applications working together over the net. The flexibility and utility is absolutely amazing, and all this for a comparatively low cost.

The system suite version of the Internet existed in the beginning in the form of CompuServe and later America Online (AOL). On AOL, you connected through a modem to their servers, and everything you'd ever need[1] was provided to you. The intention was that there was no need for you to go outside their walled garden, since they'd provide it all. And, crucially, you wouldn't need to learn to navigate any other systems. Everything was taken care of. So, why did AOL fail? It failed simply because there is no way a single corporation can provide for all the variations in needs and wishes of the population at large[2]. Also, since AOL only provided for major common requirements, smaller groups of people didn't get support for

[1] Famous last words. Almost literally.

[2] AOL didn't provide porn. That's a major reason right there.

their special requirements, exactly the thing the Internet is so good at[3].

Other examples are office suites such as Lotus Symphony, and Microsoft Office. Both were based on the idea of a cooperating suite of applications, and that it would save the user effort by not having to make different vendor's applications talk to each other. Lotus Symphony is gone, and Microsoft Office is having a hard time hanging on. Both were replaced by smaller, more specialized, applications doing a smaller part of the job better than any of these huge suites could ever do. Not only that, but the dependence of the application suite on a single platform became a major drawback, as user increasingly employ multiple devices for mobile and stationary work.

27.1 The political aspect

"Large, unified systems" is an often seen false requirement (chapter 15). The real requirement underlying this one, is the desire to move responsibility from the purchasing organization to the vendor. It has nothing to do with the functionality of the solution.

Additionally, having a single system concentrates power over the system and over its development into the hands of a few, which is often quite intentional.

27.2 The development aspect

It's often assumed, without much reasoning behind it, that large systems from one vendor are easier to develop and maintain.

Quite the opposite. It's a well-known fact, that the larger the system, the harder it becomes to add functionality. In a large systems, the effects and interactions of a change are much more widespread than in a small system.

Another problem that increases with the size of the system is the resistance to change. If some element needs to be stored or shown differently, a potentially large number of other elements need to adapt to this change. Once the change is implemented, the number of potential errors in the program is also much larger.

If we compare a system of a certain size and functionality with another group of systems with two parts, where each system is half the size, with half the functionality, then any change to this system will be at least half as complicated and with half as many problems, as long as nothing needs to change in the communication between the two parts. The chances of

[3] Providing for a large number of small groups of people and their needs is similar to "The Long Tail" as described by Chris Anderson in his book of the same name.

changes involving both parts decreases the narrower the interface is between them, that is the less information needs to be transferred between the two parts.

If you go back and look at the discussion in the section on encapsulation (chapter 24), you'll see that this reasoning exactly matches the reasoning behind object encapsulation and narrow interfaces. This is no coincidence; it's just another manifestation of the same idea.

> *So now we have reached* two *conclusions about abstraction, encapsulation, and narrow interfaces. These two conclusions tell you that we should avoid large unified systems both for technical reasons and for reasons based in the medical knowledge domain.*

27.3 The flexibility aspect

In a huge application suite, it's very hard to add or change functionality without impacting other parts of the suite. The testing requirement alone becomes almost unmanageable as the system grows, leading to very expensive development for even trivial changes, and poor testing and quality control.

If a new part of the suite contains severe errors, chances are the whole suite becomes unusable, leading to catastrophic system failures.

27.4 Document tree aspect

If we take the document tree aspect that I described previously (chapter 26) into consideration, the size of the system becomes almost irrelevant. The documents themselves are independent subtrees of the medical record of a patient, and can equally well reside in the same or in different systems. The only requirement is that each system is able to import and export these subtrees.

Chapter 28

Rules and audits

There are rules and regulations that determine what different professional healthcare roles may and may not do. For instance, only doctors are allowed to prescribe most medications, while only doctors and nurses are permitted to dispense them to patients. Only pharmacists are allowed to prepare medications, only surgeons are allowed to perform operations, and so on.

It is only too easy to assume that these permissions need to be implemented in the IT systems, such that only pharmacists have access to the system functions needed to prepare medications, and only doctors have access to the functions needed to prescribe them, but that is a dangerous path to follow. There are always situations where we have to modify these rules in order to do useful work.

The philosophical question here is if the IT system should act as an enforcer or a recorder. Should it forcibly stop users from doing things that rules and regulations prohibit, or should it let the user proceed, but carefully record what was done, so the actions could be verified or sanctioned by humans after the fact? Personally, I'm strongly leaning toward the latter, while observing, to my dismay, that most systems are heavily based on the former alternative. But doctors and nurses are inventive, so however strict the IT system is, they'll find a way to work around the barriers and corrupt it to their will.

The solution is to permit practically any and all operations as long as the user is properly authenticated, warning the user when he exceeds his nominal authority, and carefully recording the actions for any later audit. This is the way medical work was always done before computers came onto the scene, and it's the right way to work even when computers are used. Computers should help, not hinder, medical professionals in their work.

Chapter 29

General and special

The information in the medical record differs in character depending on the speciality of the doctor. Most specialities have data of a "document oriented" kind, while a few specialities are clearly process oriented.

Among the process oriented specialities, I count anesthesiology, intensive care, and medium care. Most of the rest of the specialities can be regarded as document oriented.

Among the document oriented specialities, i.e. most specialities, a certain set of data is common to all these specialities, while a second set of data is peculiar to the speciality in question.

29.1 Process oriented

Process oriented specialities have much more in common with classic relational applications, as I discuss in the section "What is the difference?" (chapter 19).

If you look at the majority of data produced and recorded in an Intensive Care Unit (ICU), it consists of a stream of measurements of arterial blood pressure, atrial pressure, blood gases, ventilation frequency and volume, urine output, drain outputs, etc. Many of these values need to be monitored for deviations outside preset limits, or are used in calculations of derived data, such as cardiac output or pulmonary shunt fractions.

Doctors and nurses also produce textual notes in the ICU, fairly regularly, but often brief. These notes cover major changes of diagnoses, and direction of therapy. They also serve as a summary of what is going on at any particular moment.

Very little of this data is suitable for other specialities, so the ICU doctors need to write a separate summary of care when the patient is discharged,

and only this summary, together with selected lab values and a medication list, is useful in the EHR outside the ICU.

Since ICUs have such different IT needs from other departments, the ICU systems tend to be built by other companies and sold separately from regular EHR systems. This sometimes results in systems that are completely isolated, so much so that even getting the summary of care written and transferred to a hospital based EHR system turns into a real problem. Ideally, the transfer of the summary with selected data should be automatic, and the search of detailed ICU data should be possible through another mechanism, if need arises.

29.2 Document oriented

Most other specialities outside Intensive Care Units (ICUs) and operating theaters, have documentation needs that are more hierarchical and "document oriented". Major exceptions to this are lab reports and demographics, which are by their nature more structured.

Many specialities have special applications and data structures peculiar to that speciality, such as vision acuity and vision area, for ophthalmologists, vascular drawings with marked stenoses for vascular surgeons, ECG storage and interpretation for cardiologists, etc. There is no need to include all these special applications in all users' copy of the EHR, especially since it would only clutter up the user interface even more than necessary.

Chapter 30

Interconnections

Connections between parts of a large system, or between independent systems, need to work, of course. There's always a desire to be able to plug in components and just have them work with no problems. To that end, a number of standards have evolved or been designed. Standards in general have quite a few problems, problems which sometimes completely overshadow any advantages they bring. Let's run through a couple of those problems *and* advantages.

Chapter 31

Presentation elements

When consulting a medical record, we have a mass of patient-related data that will need to be presented in a digestible form to the doctor or nurse. There isn't one single presentation that is useful in all clinical situations and for all medical data. Not only do we need a range of presentation patterns, we also need presentations that allow the user to easily switch to the most suitable presentation mode for a particular question or situation. In other words, we even need the right presentation of possible presentations.

But let's not get ahead of ourselves. I'll start out with a few presentation patterns that can potentially be useful.

31.1 The timeline visualization

Current medical records are largely time-based. Encounters with the health-care system are recorded in chronological order, and everything that happens to the patient is forced to fit the timeline. One could be excused for thinking that since current records are so heavily timeline oriented, that this is the most important aspect of medical history, or even of medical records in general, but then one would turn out to be very wrong. Yes, time is important, but not to the degree that it should be allowed to overshadow other aspects of the record. Even correlations in time are rarely of deciding importance.

However, it's easy to find examples when a timeline presentation really shines, for instance in visualizing side-effects of therapies, food based allergies or intolerances, and a small set of infectious diseases. But in all these cases, the diagnosis *can* be made even without a fancy timeline presentation, just by a little spatial thinking, so the question is how much we're prepared to spend to implement it. Worse, you only know the timeline is useful for a diagnosis *after* you've seen it, so to experience a sufficient

number of such happy coincidences, we would probably have to timeline the entire medical record, thereby actually making everything else so much harder to read and understand.

The really bad news, however, is that timelines *look* good. They sell systems. I fear they're here to stay.

31.2 The anatomical visualization

Another presentation form we often see in medical record related applications is the anatomical. If we reduce the record to an image of a person, then all diagnoses could be shown at a certain spot in that image. Coronary disease points at the heart, asthma at the lungs, diabetes at the pancreas, etc. Sounds good, right?

But wait... diabetes is a pancreatic problem? Really? Well, one can argue that type I diabetes (juvenile diabetes) is a pancreatic problem since it's caused by a loss of beta cells in the pancreas, but is it useful to color the pancreas red (or whatever) to show that the patient has diabetes? It could also, more plausibly, mean pancreatitis. Or pancreatic cancer.

How about diabetes type 2? That one is caused largely by a resistance to insulin in muscle tissue (I'm over simplifying for the sake of argument), so what do we point to now? All the muscles? The biceps? You see where I am going with this.

Even if we decided to point to all the muscles in the body as an illustration of diabetes 2, that still wouldn't illustrate the vascular and kidney problems often occurring in diabetes, both types. So for diabetes, you really should color the whole body. But if you get a record with a big red full body color, does that really mean diabetes? If so, which one? Or maybe sunburn?

On the other hand, a multiple trauma victim could probably very well be described this way. So it's not entirely useless, only almost.

31.3 Word clouds or tags

Yet another way to extract significance from the medical record is by word frequency lists, or word clouds. I mentioned both word and tag clouds in the section on improvements of current systems (section 12.1). These techniques could indeed alleviate some of the misery of our current text based EHR systems, but would be largely superfluous in a correctly designed EHR. Since "correctly designed" may be too much to hope for in the foreseeable future, at least the tagging system can have a place.

Disappointingly, none of the EHR systems I've seen so far has implemented either word or tag clouds, so we can't determine how much they would make the current systems more bearable.

31.4 Issue oriented

The patient has one or more "health related issues"[1], such as lower back pain, coronary heart disease, or diabetes. Even preliminary examinations, such as "unconscious for unknown reason", or "traffic accident" are valid "issues". Everything we do with that patient is related to one or more of those issues, and conversely, every issue the patient has is related to any number of actions, decisions, hypotheses, results, or outcomes. Medically speaking, that is the *only* way to view a patient.

In practice, however, things are nothing like that. Our current EHR systems do not organize information around issues. Since doctors *always* do that mentally, there is an unavoidable transformation[2] of the "view" of the patient from the flat, timeline based, organization of the EHR, to the much richer issue based mental picture the doctor needs in order to proceed. That transformation must happen in the mind of the doctor. Since the EHR cannot persist the result of the transformation, it must happen over and over again, at each new encounter.

> *Needless to say, this mental transformation isn't only mentally exhausting, it is also extremely error prone. No two transformations will look exactly the same, so the inner picture of the patient history and planning will vary from encounter to encounter. How we are ever expected to be consistent, or even correct, in treating the patient under these circumstances remains a mystery to me.*

The solution would be to link all elements of the medical record to one or more issues they are related to. That allows us to select an issue and automatically find all related prescriptions, referrals, protocols, etc. It also allows us to find out which issues are related to a particular document, a somewhat less interesting deduction, but not without merit in selected circumstances.

> *The list of the issues forms much of the organized overview we're looking for when seeing a patient, so if it is present, the whole mental transformation process can be bypassed.*

Having all documents linked to the issues they pertain to, has a number of related advantages, as I'll sum up in the following.

[1] I chose the term "issue" to represent the same thing that has been called a "problem" by others. See Laurence Weed: A Problem Oriented Medical Record (POMR). The POMR, however, utilized the "problem" concept in a much more limited way than I'm doing with the "issue" concept.

[2] When two different representations cannot be transformed exactly into one another, it's called an "impedance mismatch", a term borrowed from electronics.

Confidentiality

If we flag a particular issue as confidential, i.e. "syphilis", then all documents in the medical record that are related to "syphilis" (and only "syphilis") would remain hidden to the staff that isn't allowed to see it. This is a whole lot better than the method currently often used where entire departments get put under a "confidentiality spell".

Even if an issue is marked "confidential" and not displayed to the user, the system itself can still detect contra-indications when prescribing medicines that should not be given in the presence of the hidden issue. It's up to the policies in the system to then decide what to do with that warning, but I can think of these reasonable alternatives:

- Warn the user outright that there is a condition forming a contra-indication for the attempted prescription.
- Disallow the prescription entirely, referring to an undisclosed condition.
- Leave the prescription pending a review by a doctor assigned to the undisclosed condition, without revealing which doctor that is.
- Reveal which doctor to contact regarding the undisclosed condition.
- Give the prescribing doctor the opportunity to break the confidentiality intentionally, citing emergency reasons.
- Give the patient the chance to provide the doctor with a key, or at least a verbal permission, to break the confidentiality.

These alternatives don't entirely solve the problem, but they do at least provide reasonable ways out of the dilemma, without needlessly revealing the issue that was marked "confidential".

Contraindications

Since all medications have known contra-indications that can[3] be coded as ICD–10[4] codes or similar. Issue templates in this most marvelous of future EHR systems, are also encoded with a set of discrete ICD–10 codes, or ranges of ICD–10 codes. As both types of elements are encoded in the same coding system, it becomes quite easy and safe for the EHR system to deduce when there is a risk for a contra-indication, and warn the user accordingly.

[3] For some inscrutable reason, they don't seem to be encoded yet, as of this writing.
[4] ICD–10 is a disease classification code system maintained by the WHO.

Interestingly, if the user has employed an over broad issue, for instance "headaches" instead of the more specific "Cluster headaches", then the a warning for a contra-indication related to "migraine", a type of "headache", but not a "cluster headache", would have the effect of stimulating the user to decide to work through the record and, if possible, replace "headaches" with the more precise and limited "cluster headaches", in order to resolve the warning. In this case, the incentive to reduce bothersome warnings from the system results in a more exact definition of the issue, which must mean better tailored treatment and followup. That's a very nice synergy.

Structured input

If the record is organized into a number of "issues", then each issue can be organized into different activities or timespans. For instance, the following activities are obvious candidates if the issue is "diabetes":

- Workup: includes the criteria for diagnosis and all referrals.
- Yearly followup with lab orders and reporting to national registries.
- Diabetes coach encounters every three or six months with provision of test material, diet and glycemia followup.
- Therapy change activity, which can be activated at any encounter or between encounters if the diabetes status motivates it.

Reporting, epidemiology

Whenever the user initiates an "issue" in the record, this is potentially a reportable incident. For instance, if you initiate "gastrointestinal infection", you will be presented with the questions that are needed to report certain infections to the local infectious diseases monitoring board. By having this reporting tied in to "issues", the relevant data for the reporting is acquired together with other clinical data, eliminating double entry, and virtually guaranteeing timely and complete reporting.

Chapter 32

Coding systems

Any coding systems should be chosen according to the clinical advantages they could bring, not just to produce management statistics. This requirement is based on the observation that medical record data entered for other reasons than the direct care of the patient, tend to be intentionally or unintentionally manipulated, and thus become untrustworthy.

There are good reasons to code a medical record, such as:

- Making terms and concepts in the record machine translatable.
- Make the clinical record directly exportable for reporting and national registry purposes.

When implementing the ability to use a coding system in your EHR system, consider the forward and backward compatibility, by considering the following:

- Any item or value could be encoded using none, one, or several coding systems at the same time.
- The coding system may be updated.

32.1 Multiple coding systems

Since the actual coding system in use should not influence the basic design of the EHR system, the EHR needs to contain references to the coding system. The actual code tables used can also differ from one item to another, from one template to another, and within the same template and item, from one point in time to another, or even from one patient to another.

All this implies that *each* use of a code in the system must include a reference to the coding system used *and* the exact version of that coding system.

32.2 Updating and changes

We have to consider that recorded information in the EHR can live unchanged for a very long time. During that time, the code tables that were used while adding data to the record may very well change. The meaning of codes may differ substantially from what the meaning was when the code was first used.

To make sure that the right code table is used while reading a record, the code in the record must include a qualifier that uniquely identifies the exact code table that matches the use of the code in the record.

32.3 Finding code tables

One system of identifying a particular code table would be based on universally agreed upon naming and versioning, and this doubtlessly has its place. There are universally agreed upon tables such as ICD–10, and ATC that we can reliably refer to by name and version. SNOMED CT also belongs in this category.

But there is a formidable universe of code tables that spring up inside organizations, often for not very thoughtful reasons, but since they are used in live systems, we can't just ignore them. Neither can we prohibit vendors from using any outside standardized tables, since that will never work. See also my discussion about avoiding the need for standards (chapter 16).

Do we need a prearranged place to look for code tables? I'd argue that we don't, and we wouldn't be able to provide a reliable constant place even if we wanted to. Not everyone will ever agree on one place. Some organizations will not be let in by others. There is no reliable constant way of referring to a place. No, not even a URL, since that could also very well change when we transit from the URI/URL scheme we use today to something better in a not too distant future, for some reason none of us know today. Any scheme based on current technology has a huge risk of not working a few decennia down the road, and our current medical record data must remain useable much longer than that.

Chapter 33

Touch or not

If the EHR system is properly structured, the informational screens will be built up mainly from issue templates, which implies that most input is selected from a set of expected alternatives. Most inputs will need a standard value representing "normal", and that input should be selectable using a single simple action, like a click or a tap or double tap. If selecting a standard, but abnormal and thus less frequent, value, a simple selection from a list of some kind is appropriate. In every case, there must be an escape mechanism allowing any amount of text, since we cannot say with absolute certainty that we've covered all possible values with a list of preselected values. In most cases, there should be the ability to dictate as well.

In current EHR systems, most of the information going into the record is in the form of free text, simply because that is what these systems can handle. In this case, it's logical to use either a dictaphone, or a desktop computer with a keyboard.

In systems designed with issue templates as I'm describing, the major part of the input will be done from structured lists with selectable alternatives, which can either be managed using a mouse, or even better using a touch based platform.

Chapter 34

Security

Before discussing *how* to make the medical record secure, we must first define what we mean by "secure".

Conventionally, we view security as the combination of three aspects into a triad, which for mnemonic reasons is usually referred to as the "CIA" triad, which stands for "Confidentiality - Integrity - Availability". As is so often the case, a real implementation will always be a compromise between these three aspects to a varying degree. There is no "right" combination of these three aspects for all types of applications. You usually can only get two of the three, anyway. Let's illustrate this with a few examples.

The software that runs ATM machines is built with the "Integrity" and "Confidentiality" aspect emphasized at the cost of "Availability". If something goes wrong, the bank isn't too worried if the ATM stops working (no "availability"), as long as secrets are kept ("confidentiality"), and the correctness of bank balances ("integrity") is maintained.

The software that runs medical records, however, must always be available, always correct, and always confidential. That is an impossible triad of requirements, so something must give. Depending on application area, different aspects can be allowed to take precedence.

For instance, in emergency care we need availability most[1], integrity comes a close second, while confidentiality is a distant third. In a psychiatric outpatient setting, however, confidentiality may trump availability and integrity.

The main takeaway here is that you must make a deliberate choice of which of the three aspects in the CIA triad you prioritize in any particular

[1] Actually, one can argue that if there is one place in medicine where the medical record is needed the least, it's in emergency care. Many emergency cases are unrelated to previous illnesses and treatments, and stand on their own so to speak. For instance, if the patient has a bullet hole in the chest, you're not likely to find the cause (or the cure) in the medical records.

situation. You can't just do nothing and hope all three magically will be perfectly implemented.

34.1 Confidentiality

Confidentiality of the medical record needs to be viewed at a number of levels. On the one hand, we want to keep the entirety of the medical record safe from snooping by anyone not involved in the medical care of the patient. On the other hand, we want to enable the hiding of select parts of the medical record from caregivers that do not need to have access to them.

For instance, no part of the medical record should be available to random hackers, or government agencies without warrants. Chosen parts of the medical record should be made available to healthcare providers involved in the care of the patient, if those parts are necessary in order to improve diagnostics or therapeutics, while minimizing risks.

These two types of confidentiality are basically different. The first type, let's call it "system confidentiality" is identical to the type of confidentiality surrounding most other major systems with secrets to keep, while the second type, let's call it "medical confidentiality", has quite a number of characteristics particular to the field.

Medical confidentiality

Medical confidentiality (my term) involves keeping parts of the EHR invisible to certain groups that otherwise do have at least some access to the EHR. A patient may for instance wish that his GP doesn't know about his (the patient's) venereal disease, in particular since the GP knows (or maybe even *is*) his spouse.

The goal we're trying to achieve here is to keep a particular disease or unfortunate affliction secret from some of the viewers of the EHR. This requires that all the elements in the EHR that are connected to this problem must be clearly identified as such, so we don't suppress the visibility of too many elements. If we do, we damage the "integrity" of the medical record.

In many current EHR systems, there is no clear concept matching the disease or "unfortunate affliction" in question, so most system fake it by limiting access to parts of the records created by a particular department or provider. For instance, if the patient has schizophrenia and wishes to limit access to anything referring to that, all the records created from the department of psychiatry ends up being limited in access. This is fine (I'm being charitable, here) as long as that department is handling nothing else for this patient.

As an example of a problematic situation, assume our patient has been treated for syphilis by a urologist, and then marks the urologist's records as extra confidential, such that other specialities cannot read that record. Then what happens if the same urologist treats the patient for kidney stones? Yes, indeed, the kidney stone episode is now also confidential, possibly causing problems down the road if the patient presents with an acute abdomen at the ER.

In most current systems, the confidentiality only covers the notes part, leaving prescriptions and referrals open for everyone, as long as they can access the EHR for this patient. The reason for this is that these systems usually have no way of knowing which referrals and prescriptions belong to a particular problem area, or even which departments can be viewed as the "owner" of the information. For instance, assume that a GP refers a patient to the urologist for a suspected infection, which then turns out to be syphilis, and then the urology records are marked highly confidential. Does the referral also need to be hidden? If so, the GP won't see his own referral. If not, it looks strange to have a referral leading nowhere, with no visible response.

The underlying problem here is that these systems are designed on the presumption that the most relevant subdivision of patient's problems are according to speciality, while in reality, the most relevant subdivision is according to problem, or "issue". Additionally, the "thread of relevance" between different parts of the record is not determined by the role of the healthcare provider, but by the dependence on one piece of information on other pieces of information, as I discuss in the chapter on the document tree on page 189 and onwards.

System confidentiality

System confidentiality, or system protection, involves all the classic mechanisms, such as:

- Secure authentication.
- Secure authorization.
- Secure communication.
- Secure data at rest.

Secure authentication

This means making sure that we know the identity of the individual or system that is connecting, and it can utilize a number of techniques.

Name and password The "classic" authentication mechanism in IT systems is username and password. The advantage of this system is mainly ease of implementation. There are multiple problems with it, though.

If the user is allowed to pick the password, it will often be trivial or too easy to guess. If the password consists of any real words, even if slightly modified, it will be subject to "dictionary attacks" against the login screen. If the number of login attempts is limited to ameliorate this attack, we may make the access to the medical record too difficult for legitimate users; we've diminished availability in favor of confidentiality and integrity. Since we generally would want the balance to be the opposite, i.e. availability as a higher priority than confidentiality and integrity, this method is inadvisable.

What we can do to improve on the quality of the passwords is:

- Verify the complexity of the password as it is created.
- Allow the user to keep using the same password eternally.
- Use "password vault" software to keep complex passwords for the user.

One of many drawbacks of passwords is that they are too easy to pass from person to person. If you build an IT system that for instance only authorizes doctors to prescribe medication, doctors are certain to let nurses prescribe in their place, using the doctor's credentials. There are situations where doctors delegate prescriptions to nurses in real life, and if the IT system tries to change that behavior, it will be circumvented, and rightly so. The result is a false record, indicating that the doctor prescribed directly, with no indication of the actions of the nurse, which will inevitably lead to headaches once the record needs to be audited for some reason. If the doctor's credentials are misused, there will be no information on who misused them.

Biometrics Logging on using biometrics, if it worked right, would make it much simpler for an individual to log in. It would also make it impossible to pass on the credentials to another person. But it would also make it impossible to log in while wearing gloves (in case of fingerprints) or headlamps and glasses (in case of retinal scans), making the utility limited.

Tokens Tokens are hardware items you possess and that are hard to duplicate. An example is a key fob, or a smart card. It's easy enough to implement software and even card readers for desktop computers and tablets that can use these cards. It's so easy, in fact, that a number of organizations have dived into this without much thinking of the consequences.

One can be excused for thinking it is a great idea to use smart cards for logins to the desktop, the iPad, for opening doors, and getting free coffee

and soft drinks. But one cannot be excused for not realizing you can only do one of those things at the same time.

If removing the smart card from a desktop machine's keyboard causes your current application to close down, you have a highly secure application and a highly irate user. If that same card is used to open doors, this introduces a strange dependency between losing patient data and opening doors. To avoid crazy situations like these isn't easy. Either applications must be designed to just go into a suspended state when the card is removed, or multiple cards must be issued. This in turn raises two other problems:

1. Multiple cards are harder to control. Users may give one or more away to others, for convenience.

2. A "suspended state" is fine if only one user uses the computer or the session, but if more people do, which is often the case in healthcare, the applications need to juggle multiple independent suspended states, and that is probably too much to ask. Virtual sessions could be the solution here.

3. Even a perfectly suspended state doesn't solve all problems. If I have an iPad and a desktop, and both require the same card, I will never be able to read on one and do data entry on the other, so eradicating many of the advantages of multiple units.

Another possible solution would be proximity activation, such as with NFC cards or Bluetooth units.

If you go with smart cards and use a full PKI, you have another set of problems to contend with as a consequence of revocation systems. If a user loses his card, you can revoke it through the PKI mechanism. The problem occurs if the card is found again. Since there is no un-revocation mechanism, you have to issue a brand new card then.

An alternative would be to only deactivate that card in the EHR systems while not issuing a full revocation, allowing you to reactivate it if it is found again. But if you can do that, there was no need for a PKI based card in the first place. You could just as well issue your own institution based cards, saving quite a bit of money in the process.

Database security

In most systems, the database as such contains plain unprotected data. The assumption is that this data can only be reached through a number of processes that are well protected, so there should be no road to the sensitive data store. That has proven again and again to be wrong. There is *always* a way in, however well you protect it.

The solution, which should and almost certainly will be mandated in health-care, is to encrypt data "at rest". That encryption should be strong enough so that even if unauthorized persons got hold of the database[2], nothing about the patients in that database can be revealed.

Loss of databases have happened as programmers, consultants, doctors, or management, have put data on regular USB keys and lost those, or lost laptops, or even had criminals walk out with entire servers from data centers. The problem with these types of losses are that they're not only frequent, but that the sheer amount of data compromised each time is enormous. As to numbers, the medical record compromised in the US alone, in the period 2005–2013 involves almost 30 million medical records in more than 1000 breaches, and these are only the published ones, and only the breaches reported from healthcare organizations. Other organizations also have breaches which alone or in conjunction with other breaches, reveal sensitive data about patients.

The system as gatekeeper

With all these methods of authentication and authorization comes the question of how actively the medical record system should be in enforcing the authorization. In other words, if a nurse is not allowed to prescribe narcotics, should the system disallow the nurse that prescription or not? The simple answer would be "yes", since that's what the law says. The more complicated answer is, um, more complicated.

Traditionally, doctors have allowed nurses to do many things that formally should only be done by a doctor, or in very close supervision by a doctor. When a doctor lets a nurse administer morphine without a formal prescription on paper, it is often done because the doctor can't leave the operating theater or is otherwise unavailable. You can't just leave a patient in pain because of formalities.

If we program the computer system to not allow a nurse to perform these borderline admissible actions, we'll have one of three things happening:

- The patient will be left suffering.
- The nurse will administer the morphine without recording it anywhere, increasing the risk of interactions and accidental overdoses.
- The doctor will let the nurse use the doctor's credentials, falsifying the record in the process.

[2]Entire databases have been lost so many times it's not even funny. For some disturbing statistics, see https://www.privacyrights.org/data-breach/new

None of the above outcomes are even remotely acceptable. To avoid them altogether, you will have to let the system follow actual practice, so the nurse can prescribe the morphine, and have a solid record of what was actually done. The system should let the nurse enter a motivation for the action, and let the doctor authorize the action after the fact. If something goes wrong, the record can be used to find out what went wrong, and who is to blame. That is exactly what we always did before the introduction of computerized medical records, and we should be allowed to continue that practice.

> *The moral of the story: it is not the role of the electronic healthcare record system to enforce the law or policies, only to record and advise. If it places unreasonable restrictions on clinical work it should, and will, be circumvented.*

Reduce the attack surface

It's hard, and becoming increasingly harder, to protect medical data from snoops. As we now know[3], there's a whole array of extremely capable organizations that are after our data. They don't have the interest of our patients at heart, and our chances of keeping them out are near zero. We must assume that our ability to keep secrets is close to nonexistent, and must act accordingly. What we need to do is reduce the attack surface[4].

There are three ways to handle this situation:

1. Go back to paper and/or entirely isolated systems[5].
2. Avoid documenting sensitive data at all, if possible.
3. Avoid creating potentially harmful data we don't really need.

Since it's extremely hard to know beforehand which data elements can be "potentially harmful" in conjunction with other data, in future situations we cannot foresee, we have to regard *any* data on the patient as potentially harmful.

[3] As I write this, the Snowden revelations are in their sixth month, and there doesn't seem to be an end in sight yet.

[4] This term is usually used to indicate the amount of code that can be the object of an attack, but I'm using it to mean the amount of medical information about a subject that can potentially be compromised. Same thing, if you think about it. Or even if you don't.

[5] We have also seen from the Snowden revelations that it is much harder to protect even non-connected systems than we thought. So if you handle high-value "targets" as patients, no IT system, however isolated, can be regarded as safe.

Only the essentials

We should avoid adding in any inessential information into the medical record, however innocent it may seem. Any really sensitive information should be entered into fields for this purpose, which in turn should be extra protected[6] or limited in distribution[7]. That is what we should do, but I don't really think we can (see the footnotes), which leaves us in an untenable situation.

There's an interesting corollary, at least if you accept my conclusion that the current generation of medical record systems do not help us much in delivering better healthcare, but only in improving the management view of the production process, namely:

> *Assuming the current generation of unhelpful EHR systems, the overall effect of entering extensive and sensitive data on the patient into the electronic healthcare record and national registries has a negative effect on the patient's privacy and security that may well* exceed *the net positive effect on his health and well-being. The only way to change this poor outcome is to increase the usefulness of the EHR, while reducing the amount of personal information entered, and at the same time improving the security of the EHR.*

Reduce at the source

If we produce data for public databases such as national registries, we should anonymize the data at the source. What we see today, in general, is that identifiable data is transmitted to the registry, then only anonymized during extraction of data. Three bad assumptions underlie this design:

- That you need the patient identity to correlate records and avoid duplicates.
- That you can protect the patient identity at the registry.
- That the information in the registry isn't damaging to the patient.

[6] How we're supposed to do that, is another question entirely. And if we could protect it sufficiently, shouldn't we do the same thing to the rest of the record?

[7] Limiting the distribution of some data but not others, introduces the problem of compromised integrity of the medical record. It is possible to write up the record in such a way that the remaining, "unlimited" data still is correct and non-misleading, but it is almost certainly too much to ask of doctors and nurses to be aware of what exactly needs to be done to achieve this. Only a very rigidly structured medical record system could have a chance of producing safe data this way.

Need for patient identity

A national registry needs to be built in such a way that we can identify records belonging to the same patient, in order to allow the identification of duplicate reports, and to track the evolution of the disease per patient. Put in another way, the need is to have a mechanism whereby we can confidently link different reports belonging to the same patient, or duplicate reports on the same patient. This does *not* mean that we need the *actual* identity of the patient we're tracking.

Ability to protect the identity

The easy, and wrong, way of creating a pseudo identifier is to include a unique identifier per patient in the national registry, as is usually done today.

If that unique identifier is the official identifier of the patient (such as a national number or personal number), or a derived number (such as a hash), makes no big difference. If the unique identifier is constant for any particular person, it doesn't hide the identity, at least not for long. It only adds a thin layer of fairly easily crackable obfuscation.

Many current systems use hash functions to derive an identifier, but this is totally ineffective. If the hash function is known, it's trivial to create a lookup table of every possible real personal identifier number and its corresponding hash value, then use that table to find the real number easily. This is called a "dictionary attack". A lookup table for all possible personal numbers in Sweden would use up a smaller capacity USB stick, no more. For the specific discussion see appendix on dictionary attacks, page 235.

If you try to keep the hash function secret, you will fail. By necessity, you'll need to communicate the function in source or executable form to hundreds, if not thousands, of developers creating compatible applications, so intentional or accidental leaks are unavoidable. Worse, once leaked you can't change the function, since the registry database depends on the hash values of old and new records.

Now, just for the sake of argument, let's assume you invent a good hash function, or even better, use a Message Authentication Code (MAC[8]), and the function (or in the case of a MAC, the secret key) doesn't leak, you can *still* fairly easily identify the records belonging to a particular patient, if you can only correlate a single one of them.

Example: assume Norbert Y, an AIDS patient who has a series of records in the AIDS national registry. One day he goes to his GP for a visit. The GP sends out a record for the national registry. Now, anyone listening in who sees that record go out and has access to the appointments calendar

[8] http://en.wikipedia.org/wiki/Message_authentication_code

for the doctor, now knows what unique identifier this patient has. And that identifier now lets the eavesdropper find *all records, past and future, in any registry or database* for that patient. And there is no way to recode that identifier to something else, if you find out what happened. Now this is a huge downer.

Now, there is a fairly simple way to achieve good identification protection without the drawbacks mentioned. I'll describe the design of such a system in the appendix on registry anonymizing on page 237.

Innocence of data

The argument is often brought forward that the data in the national registry is innocent, it's "nothing to be ashamed of". But *no* data is innocent in all contexts, and this data can be *very* damaging to the patient if used in conjunction with data from other sources. There is no predicting how it will be used and by whom, if it is compromised.

34.2 Integrity

The integrity of the medical record means that it should be consistent, complete, and truthful. The introduction of new information, or change of existing information, should be carefully controlled.

But to the classic definition of integrity in medical records, I would add: all decisions recorded in the medical record should be based on clearly defined sources, including reports, results, clinical examinations and signs, *and source knowledge used in drawing conclusions and making decisions*. This implies that all source data is somehow identifiable as distinct chunks of text or other data, and also addressable as such, that is, each have a unique identifier.

It turns out that the only way to achieve this simply and reliably is by not allowing updates or deletes in the medical record, only additions of complete and verifiable chunks of data. I discuss this as part of the description of the document tree on page 189.

34.3 Availability

Availability of the medical record is often disappointingly poor. In many countries, entire populations are left without any medical data at all for hours or even days as systems go down. I find it amazing how poorly these systems are architected, and how negligent the vendors and the healthcare

providers are in testing these systems. The whole setup reeks of cheapskates and unfounded optimism.[9]

[9]Oh, sorry, got carried away there. But contemplate this: the province I worked in for a while upgraded their servers, since the old one couldn't carry the load. The old one was then set up as a backup in case the new one went down. When I asked the system admin if they've ever tested switching over, he said "no, we can't disrupt the hospital just for a test". When I asked why he thought it could work and take the load, he said "of course, we used to use that server before". When I then said: "yes, but you replaced it because it couldn't take the load...", I only got an uncomprehending glare back. Is it just me, or do we have a problem here?

Appendices

App. A: Attacking hashed personal numbers

If the hash function is known, how large is a full dictionary of all personal numbers in Sweden with matching hash value?

Swedish personal numbers are built up as YYYYMMDDNNNC, where YYYY represents the year of birth, MM the month (00–12), NNN a number, unique for each person born that day, with males having odd numbers as last digit, and females even, and finally with C being a check digit.

If we take a span of 100 years, 12 months per year, 30 days per month, and 1000 numbers per day (discard the check digit), we'll arrive at 36 million possible numbers. If we assume that this number takes 8 bytes of memory, and a 128-bit hash takes 16 bytes, then we're pretty sure it all fits in 32 bytes, right? There's even breathing room left in there.

If we then use the hash value as a clustered index[10], we're going to get *really* fast lookups with a total data storage space of a bit over 1 GB. We're talking milliseconds here. Pretty amazing, huh?

So *that* is what a simple hash brings you: nothing.

[10] In a clustered index, the part of the data you are searching on is arranged in the order it is looked up, so you don't need an extra file or array just for that lookup. (Yes, I realize this explanation is oversimplified, bordering on being wrong, but maybe it gives an idea of what the intention is with a clustered index, namely one less look up and some significant reduction of size.)

App. C: Registry anonymizing

The requirements on a solution for reporting anonymized clinical data to an outside registry can be summarized as follows:

- The identifier of the data at the registry should bear no relationship to the identity of the patient.
- The identifier of the data at the registry should be different for each instance of data, even though the data may come from the same patient. We can call this "forward anonymizing" (equivalent to the concept of "perfect forward secrecy" in communications).
- Different records in the registry should be able to be matched by a third party as belonging to the same patient without giving away any clue that would allow the registry to determine such a match of future records without help from the third party.
- The third ("matching") party mentioned above, should not be able to deduct what type of clinical data belong to the identity records it is matching.
- Different registries should be able to cross match for the same patients if using the same third party identity matcher, or with several cooperating third party matchers, still without revealing to the third party which type of clinical data is involved.
- No single cryptographic key should exist that would compromise past and future encrypted data in either the registries or the third party services.

Public key cryptography

To understand the explanation, you first need to understand what public key cryptography is. Not the details, mind you, but the general idea[11].

There is a way to create a pair of keys, entirely different from each other, but still related in such a way that when something is encrypted with one of the keys, it can only be decrypted with the other key of the pair. It can't even be decrypted by the same key it was encrypted with, only with the other.

Imagine you have two people who want to communicate. As we usually do in cryptography, we call them Alice and Bob, which gives us the convenient notation A and B for the parties.

First, Bob uses a public key crypto system (PKCS) to create a key pair, then sends one of the two keys to Alice. Now Alice can use the key she received to encrypt a message, then send the encrypted message to Bob. Bob can then use the other key from the pair to decrypt that message. The shorthand notation for this is:

$$A \rightarrow B : \{M\}_{PK_{\text{pub}}(B)}$$

The meaning of the different parts of this expression are summarized below (Table 34.1).

When Bob created his key pair, he kept one of the keys for himself and sent the other key to Alice, so she could use that key to encrypt messages that only Bob could decrypt. When you think of it, you realize that Bob could actually publish the key he sent to Alice, so *anyone* could pick it up and use it to send messages to Bob that only Bob could decrypt. The key that can (and maybe will) be made public is called the "public key", while the key he keeps private is called the "private key". Any one of the keys in a key pair can be chosen to be public or private; there's no qualitative difference between them. But once you've chosen to publish one of them, the other must be kept private.

Now this is really confusing, since the entire method is called "public key cryptography", while "private key cryptography" refers to an entirely different kind of cryptography where there is just one key shared between the parties. To make a clear distinction, I'll use notation as given below

[11] In this explanation, I may be upsetting the cryptographically well versed among the readers by simplifying how public key crypto is usually implemented. In reality, when sending a message, it is first encrypted using a random symmetric key, which in turn is encrypted with a public key system. This is done for performance reasons, since public key crypto is *really* slow. That simplification makes no difference in the design description here, and once you decide to implement it, you're going to ask someone who knows this stuff, anyway. Right? Right?!? Oh, no... not again... I can see it in your eyes...

Table 34.1: Security protocol notation

Expression	Meaning
$A \rightarrow B$	Alice(A) sends something to Bob(B)
$A \rightarrow B : M$	Alice(A) sends message M to Bob(B)
M	The plain text message
$\{M\}_{PK_{\text{pub}}(B)}$	Message is encrypted using the public key belonging to Bob(B)
$M_{E(B)}$	The same as previous, encrypted using public key belonging to Bob(B), but a simpler form
$\{X, Y\}$	Two or more elements inside curly braces denotes that they're bundled together as a block. If there is no subscript, there is no encryption of that block.

(Table 34.2). A "public key pair" consists of a "private key" and a "public key". Terrible, I know, but that's how these things are.

Table 34.2: Key notation

Expression	Meaning
PK_{priv}	Private key of a public key pair
PK_{pub}	Public key of a public key pair
$PK_{\text{priv}}(B)$	Bob's private key (Bob created a pair and kept this one key for himself)
$PK_{\text{pub}}(A)$	When Alice made a pair of keys, she made one of them public. This is the one.

Encryption

You can use public key cryptography for two purposes: encrypting messages and signing messages.

Encryption works as I described above in the introduction to public key cryptography. That is, good old Alice creates a key pair and makes one of the keys public, either by sending it to anyone she wants to receive messages from, or publishing it in a repository or catalog somewhere where anyone can look it up and retrieve it[12].

[12] Yes, yes, I know, there's this CA and PKI thing, and I'll get to it in due course, see page 243.

So anyone who feels like sending Alice a message, can go to the public repository to retrieve her public key $PK_{\text{pub}}(A)$, use it to encrypt the message, then send the encrypted message to her. Let's call this anyone Charles(C), and let's call the plain text message M, resulting in the following expression describing what I just said, but in much shorter (and clearer?) form:

$$C \rightarrow A : \{M\}_{PK_{\text{pub}}(A)} = M_{E(A)}$$

The receiver, Alice, can decrypt this message back to its original form using the following process, where her private key plays a crucial role:

$$M_{E(A)} \overset{PK_{\text{priv}}(A)}{\longrightarrow} M$$

Digital signatures

Digital signatures are in essence not much more than the public key encryption process I described above used in reverse.

In encryption, anyone can create an encrypted message for Alice by using her publicly available key $PK_{\text{pub}}(A)$. Only Alice can decrypt that message, since only Alice has access to her private key $PK_{\text{priv}}(A)$.

Now, if Alice instead encrypts a message using her private key $PK_{\text{priv}}(A)$, it follows that *anyone* can decrypt it using her publicly available public key $PK_{\text{pub}}(A)$. The interesting thing is that being able to decrypt it confirms that the message *must* have been encrypted by Alice, since nobody else has her private key. Remember what I said in the beginning of this section, that anything you encrypt with one of the keys in the pair can only be decrypted by the other key in the pair.

So a signature could theoretically simply consist of an encrypted form of the message, but encrypted using a private key, so that it can be decrypted by anyone with access to the matching public key. However, simply encrypting the very document we want to sign doesn't work for two reasons:

- If you don't know what the message should look like decrypted, you can't be sure if the decrypted form is right. The problem here is that you can decrypt *anything* with any key, there will always be a result of some kind, but it will never be the same as the originally encrypted form unless you have the right key. So the plain text form must be known in some way first.

- The performance hit would be prohibitive for anything but the most trivial documents.

The solution is to first derive a number from the document using a "hash function". You can imagine this function as something that would add up all the values of the letters into a single sum, but much more complicated than this. Any document you put through a hash function comes out as a large number. The same document always results in the same number, but different documents always result in different numbers.

So the way we're applying a digital signature is as follows. First we derive a hash number through a hash function. The function I'm using in this example is called "SHA"[13]. Assuming we want Alice to sign a message M, this can be expressed as:

$$SHA(M) = M_H$$

Now we encrypt this number M_H with Alice's private key $PK_{\text{priv}}(A)$ to get the encrypted version:

$$M_H \overset{PK_{\text{priv}}(A)}{\longrightarrow} \{M_H\}_{PK_{\text{priv}}(A)}$$

To simplify things, we'll call the right side of the above expression just $M_{S(A)}$ indicating the signature of the message M using Alice's private key, like so:

$$\{M_H\}_{PK_{\text{priv}}(A)} = M_{S(A)}$$

That makes the entirety of the operation look much simpler:

$$M_H \overset{PK_{\text{priv}}(A)}{\longrightarrow} M_{S(A)}$$

Then we send both the original message M and the digital signature $M_{S(A)}$ to Bob, who can verify that signature using Alice's public key, which Bob can get from just about anywhere. Bob first takes message M and calculates the hash value from it, just as Alice did, using SHA:

$$SHA(M) = M_H$$

Bob then decrypts the signature using Alice's public key:

$$M_{S(A)} \overset{PK_{\text{pub}}(A)}{\longrightarrow} X$$

The M_H value from the first of these two calculations must match the X calculated in the second.

[13] Stands for "Secure Hash Algorithm". It comes in different variants depending on length of the produced hash result, but that is not important for this discussion of principles.

$$M_H = X$$

If they don't match, either the message signature $M_{S(A)}$ wasn't created using Alice's private key, *or* the message M isn't exactly the same as the one used by Alice to create the signature $M_{S(A)}$. In either case, the signature isn't valid.

Combined use

Often we use encryption and signatures in combination, so that the message M is encrypted and accompanied by a signature. You can decide if you apply a digital signature on the plain text message or on an encrypted form of the plain text message, and both variants have their advantages, but all general forms of the combined use apply a digital signature on the plain text version of the message[14]. In this case, you create a signature on the plain text message, encrypt the message, then send both parts together in a package. There are two ways of sending a package like this:

- Signature and encrypted message as two independent parts of a message. This is called a "detached signature". Note that the digital signature is separate, but cannot be verified without decrypting the encrypted part as well.
- The plain text message is packaged together with the signature in one block, then the entirety of that block is encrypted. In this case, the digital signature cannot even be isolated without first decrypting the package. The advantage is that only the ultimate recipient will be able to determine who signed the message. There are cases where this is a desirable trait.

In the first case, when delivering a detached signature with the encrypted document, assuming Alice signs and encrypts for Bob, we can denote it as follows:

$$M \xrightarrow{SHA} M_H$$

$$M_H \xrightarrow{PK_{\text{priv}}(A)} M_{S(A)}$$

[14] In most real implementations, the key pair used for encryption is different from the key pair used for signatures. This significantly increases the security of the system, but is an unnecessary complication for this basic introduction, so I'm leaving that factor out of it. Here, I'll assume a single key pair (two keys) for each party, while in reality there would be two key pairs (four keys) for each party.

Now we have the digital signature, so we encrypt the message using Bob's public key:

$$M \xrightarrow{PK_{\text{pub}}(B)} M_{E(B)}$$

Finally, we package these two parts together in a message:

$$M_{S(A)} + M_{E(B)} \rightarrow \{M_{S(A)}, M_{E(B)}\}$$

... which Alice then sends to Bob:

$$A \rightarrow B : \{M_{S(A)}, M_{E(B)}\}$$

The second case, where the entire block, message and signature, is encrypted looks like this, where we first create the signature as above:

$$M \xrightarrow{SHA} M_H$$

$$M_H \xrightarrow{PK_{\text{priv}}(A)} M_{S(A)}$$

But now we reverse the actions, first packaging the plain text message and the signature together into a package:

$$M + M_{S(A)} \rightarrow \{M, M_{S(A)}\}$$

In the next step, we encrypt the entire package using Bob's public key:

$$\{M, M_{S(A)}\} \xrightarrow{PK_{\text{pub}}(B)} \{M, M_{S(A)}\}_{E(B)}$$

As you can see, the signature $M_{S(A)}$ is hidden inside the encrypted package and will only become visible after the receiver (Bob) decrypts the outer "envelope".

Certificate authorities

In the preceding text, I glossed over the problem of finding the public key for any communicating partner with the glib "you can find it anywhere". The problem here is that you can't be sure it's the right key. If you need to verify that Alice signed a document of some sort, you need to make sure that the public key you use to verify it actually belongs to Alice.

There are several ways you could do this. The first and most obvious way is if you get the key from Alice personally and from that moment on keep it in a safe place.

The second way is if you get it from another person who you trust, who guarantees that he or she got it from Alice.

None of the above methods are very scalable[15], so a third method was invented whereby a commercial institution accepts public keys from users, verifies their identity either in person or by documentation in some way, then *signs* the public key with their own private key.

This commercial institution is called a "Certificate Authority" (CA), and the signed public keys it provides are called "Certificates".

There's one problem left here, namely: how do you know that the CA actually signed the certificate and not Random Hacker[16]? To be able to verify that it was the CA that signed the certificate, you have to receive the CA's public key from a trustworthy source. At this point, it may seem as turtles all the way down, but it isn't, really. It's much easier to make sure, just once, to get the CA's public key from the CA itself, or from an intermediary, and then use that to verify all other public keys you get. This method hasn't really changed the problem's nature, just reduced it to manageable proportions, and that's all it takes.

Basic design

After that quick introduction to public key crypto systems, we can finally outline how a working anonymizing system could work. In what follows, I use the symbols in the table (Table 34.3) in the different expressions.

Table 34.3: Symbols used in the system

Symbol	Meaning
I	Patient identifying data, could be name, personal number, etc
C	Clinical data without personal identification
R	Registry, the organization holding clinical data for research
T	Third party identity manager, holding the identifying data
D	Doctor or healthcare provider, creates the clinical data
N	A nonce ("number used once"), a random number

[15] There's an exception: Pretty Good Privacy (PGP) is a system where the person-to-person trust network is implemented in a scalable way. Worth looking into. You have my blessing.

[16] Can you *imagine* giving that name to your child?

Anonymizing the data

Initially, the doctor collects clinical data on a known patient, so the doctor holds the following data in his system:

$$D : \{I, C\}$$

The doctor's computer creates a nonce N, which is a random number, and creates two data packages, one containing the clinical data and the other containing the identifying data. Both packages also contain the nonce, the same value in both packages:

$$\{I, C\} \overset{\text{random}}{\longrightarrow} \{I, N\}, \{C, N\}$$

The doctor then proceeds to encrypt the identifying data and nonce with the public key belonging to the third party identifying organization T, and encrypts the clinical data and nonce with the public key belonging to the registry R:

$$\{I, N\} \overset{PK_{\text{pub}}(T)}{\longrightarrow} \{I, N\}_{E(T)}$$

$$\{C, N\} \overset{PK_{\text{pub}}(R)}{\longrightarrow} \{C, N\}_{E(R)}$$

It's important to note that in each pair of encrypted messages, the nonce N is the same, connecting the two parts together, while for every new set of clinical data and identifying data, a new nonce is generated, so that there is no connection between one package with clinical data and another, even if the patient is the same.

Finally, each of these packages are sent off to their respective receivers:

$$D \to T : \{I, N\}_{E(T)}$$

$$D \to R : \{C, N\}_{E(R)}$$

Clearly, each of the receivers can decrypt their respective messages:

$$T : \{I, N\}_{E(T)} \overset{PK_{\text{priv}}(T)}{\longrightarrow} \{I, N\}$$

$$R : \{C, N\}_{E(R)} \overset{PK_{\text{priv}}(R)}{\longrightarrow} \{C, N\}$$

Matching data

When the registry R needs to analyze data, they need to match up clinical data records belonging to the same patient into lists, without needing to know exactly *who* the patient is. This is done by collecting the nonces N from all the relevant clinical records and sending them over to the third party managing the identity store T. The identity store then sends back nonces arranged in groups, where each group corresponds to one individual patient. No information about that patient, except this grouping, is sent across.

Expressing this as we're slowly getting used to, looks like this if we're sending n number of nonces N from the registry to the identity store:

$$R \rightarrow I : \{N_1, N_2, N_3, ..., N_n\}$$

The identity store then groups the nonces together in groups, each group representing one patient, and sends that back:

$$I \rightarrow R : \{N_{a1}, ..., N_{ap}\}, \{N_{b1}, ..., N_{bq}\}, \{N_{c1}, ..., N_{cr}\}, ..., \{N_{m1}, ..., N_{ms}\}$$

... where a, b, c through m are different patients, and they each have p, q, r, and s number of different nonces belonging to them.

The system results in a simple protocol eliminating any personal identification data from the registry while maintaining the ability to perform any population based studies you can imagine.

It's important to avoid having any collusion between the registries and the identity service T, so I propose to set up the latter as a cooperative effort with judicial and patient representatives. Under no circumstances should it be under the same management as the registries.

Threats and compensations

There are a number of threats to this model that needs to be considered and handled. All of them introduce complexity that make the system a bit harder to understand, but none of them change the ultimate use and utility of the system.

Connection D to T

In the outline of the system, I let the doctor's system send the clinical data to the registry, and at the same time the identifying data to the third party identity service. This causes some problems:

- More complicated communication structure, means more frequent failures.
- If the identity service provider changes, too many client systems need updating.
- The identity service may deduce what kind of clinical data is being reported simply from the fact that it knows which organization or health provider is sending in the identifying data.

This leads to a modification, namely that the doctors system sends *both* packages to the registry provider R, but since the part with the identity data is encrypted in such a way that only the identity service T can decrypt it, the registry provider can do nothing with this part of the package except forward it to the identity service, so no new knowledge is divulged to the registry holder.

So the transmission now looks like this. First both blocks of data are sent to the registry, then the registry sends the identifying data to the identity service:

$$D \to R : \{C, N\}_{E(R)}, \{I, N\}_{E(T)}$$

$$R \to T : \{I, N\}_{E(T)}$$

There is no need for the registry holder to immediately forward that data. It can be kept in storage until the next analysis of the database requires matching and that particular block of data is involved.

Connection R to T

If the registry R sends the identifying data encrypted to the identity provider T, then knowing the registry it is coming from, the identity provider could deduce what the clinical data it never saw was about. For instance, if the identity provider receives a block of encrypted identity data from the HIV register, its staff could conclude that all the persons in that data, which they can read after decryption, have HIV.

To compensate for this risk, there are two methods, flooding or indirection.

Flooding

We could simply overwhelm the identity service with a huge amount of false requests diluting the real information among it. The extreme would be if we sent fake encrypted identity blocks with nonces for the entire population

at once, every time. There would be no way the identity provider could know which of these correspond to real nonces, and which were fabricated by the registry as confounding filling.

It's easy enough for a provider to produce these fake packages, since nonces are just random numbers, personal identifiers are sequential, and finally, the encryption key for the identity provider is public.

This method isn't as wasteful as it may sound, since both the nonces and the identity data is pretty small, as is the population, in computer terms.

Proxy

Another method is to introduce a proxy service Y between the registry R and the third party identity service T, such that the registry sends identity blocks to the proxy and the proxy sends them to the identity service provider, see figure 34.1. If we start from the doctor D sending to the registry R, then go on to the registry R sending to the proxy Y, and the proxy Y sending to the identity service T, it looks like this, taken sequentially:

$$D \to R : \{C, N\}_{E(R)}, \{I, N\}_{E(T)}$$

$$R \to Y : \{I, N\}_{E(T)}$$

$$Y \to T : \{I, N\}_{E(T)}$$

The proxy service Y knows where the data comes from, which registry, and thus which diseases can be involved, but does not know the identity of the patient.

The identity service T receives the data from a proxy Y that is used by all the registries, so it can't conclude which diseases were reported for this patient.

To increase the difficulty level even further for snoops at any of these organizations, the proxy should collect data over a certain time period, perhaps 24 hours, before randomly mixing the records so they're not sequenced chronologically, then sending them on to the identity service.

You could also introduce flooding into the proxy system to make it even better. Perhaps use flooding with fake records in case the number of records falls below a preset level which would otherwise make correlations feasible for a snooper at the identity service.

When analyzing registry data, it's also advantageous to both send and receive the lists of nonces through a proxy, so as to hide from the identity provider which clinical registries are involved. See figures 34.2 and 34.3.

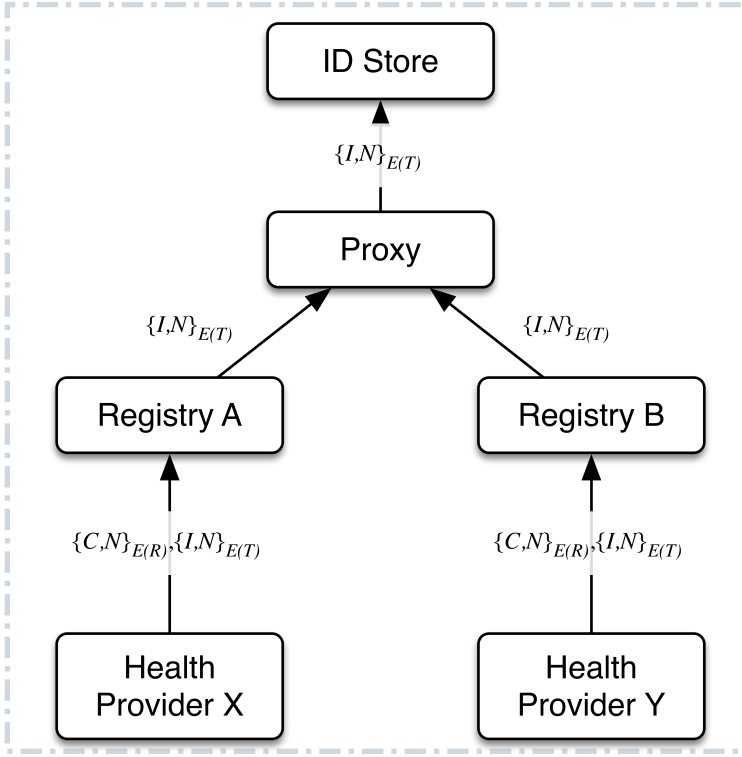


Figure 34.1: Sending data to registry and identity service

When the nazis come

A common, and very real, concern with these systems is if an enemy, or corrupt officials, take over the country. What happens to our personal data and any possibly politically sensitive diseases we have?

If all the identity records are kept separate from clinical data, it doesn't guarantee we can keep the data safe, but the problem is at least tractable. There are a number of ways we can design a system that causes an almost instant destruction of these records. The technology is there, but it's up to civil rights organizations to design a suitable policy framework around it. With the registry data being kept as it is today[17], there's no hope of keeping our sensitive clinical data from being trampled and misused.

[17] If we'd kept them in a shoebox with a string around it, it wouldn't be worse.

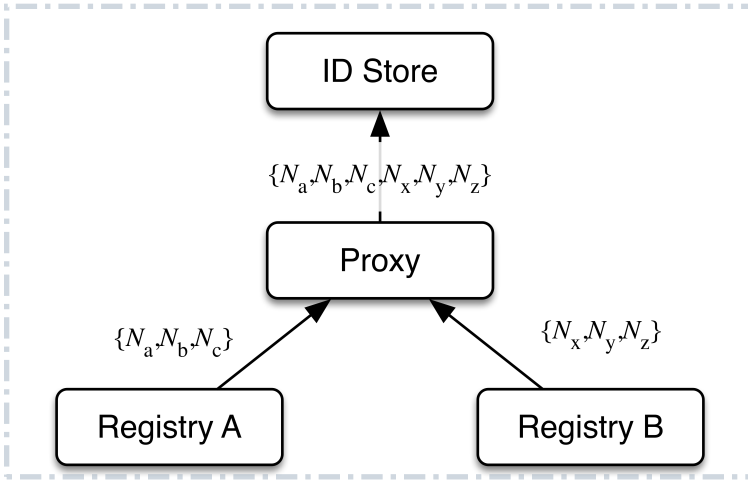


Figure 34.2: Sending lists of nonces for grouping

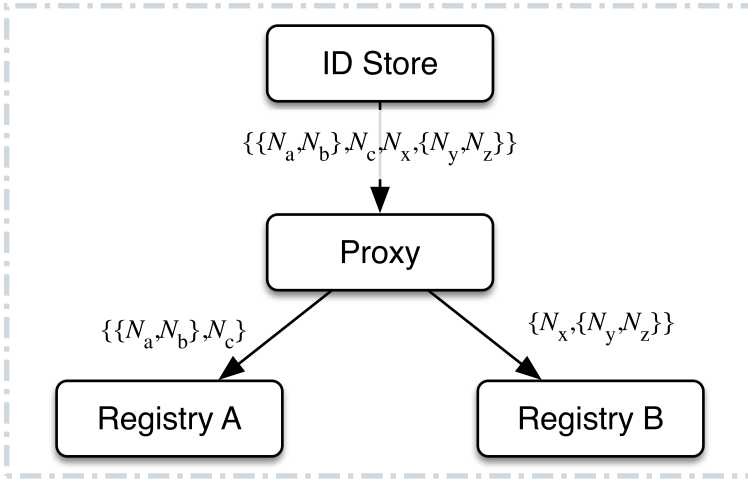


Figure 34.3: Receiving grouped nonces back from the identity service

App. E: Document tree design

Whenever we as doctors write a summary or a reply to a referral, we always base our conclusions on other documents. It could be lab reports, radiology reports, patient history, clinical examinations, or other replies to referrals. The conclusions we reach are only as valid as those source documents are, with the added ingredient of our own competence and cognition. When we sign off on a conclusion, we do that on the condition that the underlying documents were produced in a similarly considered and reliable way. If any of the underlying documents turn out to be false, erroneous, or ill considered, our conclusion also becomes, at least potentially, invalid. All this needs to be reflected in the way we manage documentation, not just having all the documents in an EHR system thrown in a heap, but by having explicit dependencies of some documents on others.

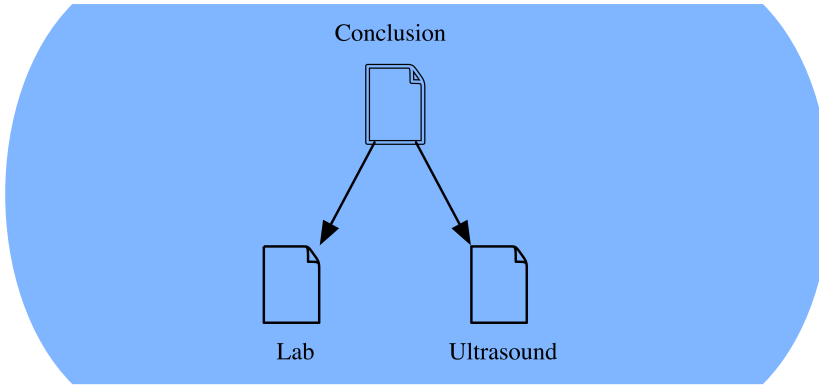


Figure 34.4: A single note, dependent on two underlying documents.

If we look at a single note in the medical record that contains a conclusion made by a doctor, based on a lab report and an ultrasound protocol, this relationship can be viewed as in figure 34.4. The "conclusion" note itself contains a text where the doctor describes what the lab report and the

ultrasound protocol mean, which diagnosis is made, and recommendations for further workup or treatment. The note with that text also contains links to the mentioned documents, represented as the downwards pointing arrows in the diagram. These links make the dependency of the conclusions in the note explicitly conditional on the content of the two underlying documents (lab and ultrasound).

As each level of medical specialist refers to a number of base documents, the specialist provides a summary at a higher level than the parts that form the basis for the new summary. The higher level summary of the lower level documents form the same kind of abstraction I described in the chapter on object oriented knowledge management, see page 181.

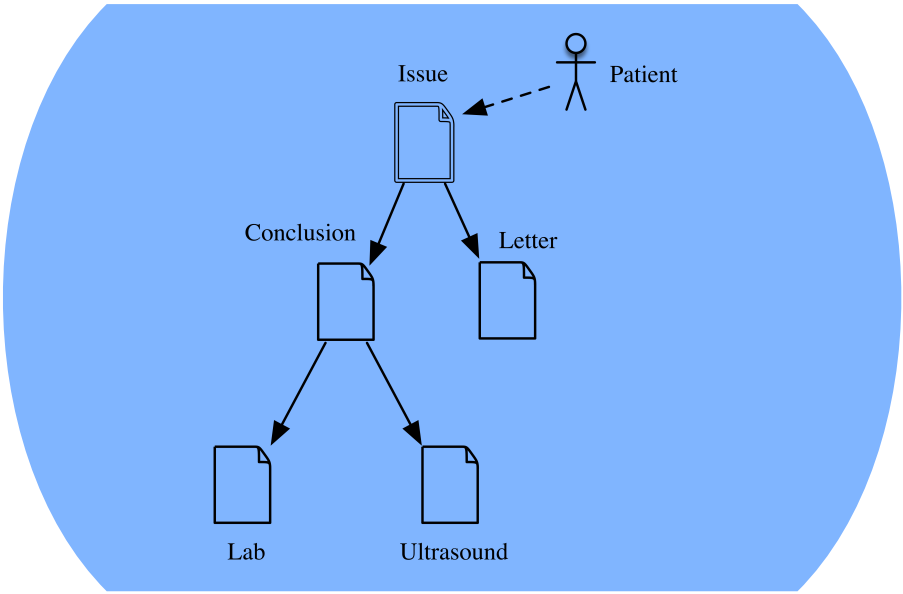


Figure 34.5: Patient can have access to the root document, the issue.

This increasing level of abstraction building on lower levels of abstraction extend all the way out to the patient, who gets the ultimate summary of earlier and more detailed summaries. In figure 34.5 I've shown that if the patient has access to the root element, the issue itself, he automatically has access to all underlying documents tied to this issue.

If a particular element in the document tree is still on the attention list and not linked to the issue (see page 193 for the discussion on the attention list), and the patient has direct access only to the issue itself ("liver problem" in figure 34.6), the patient will not see the not yet handled element since

it is not part of the tree the patient has access to. If your policy says that patients should only see parts of the record that have been seen and handled by a doctor, this is the perfect solution.

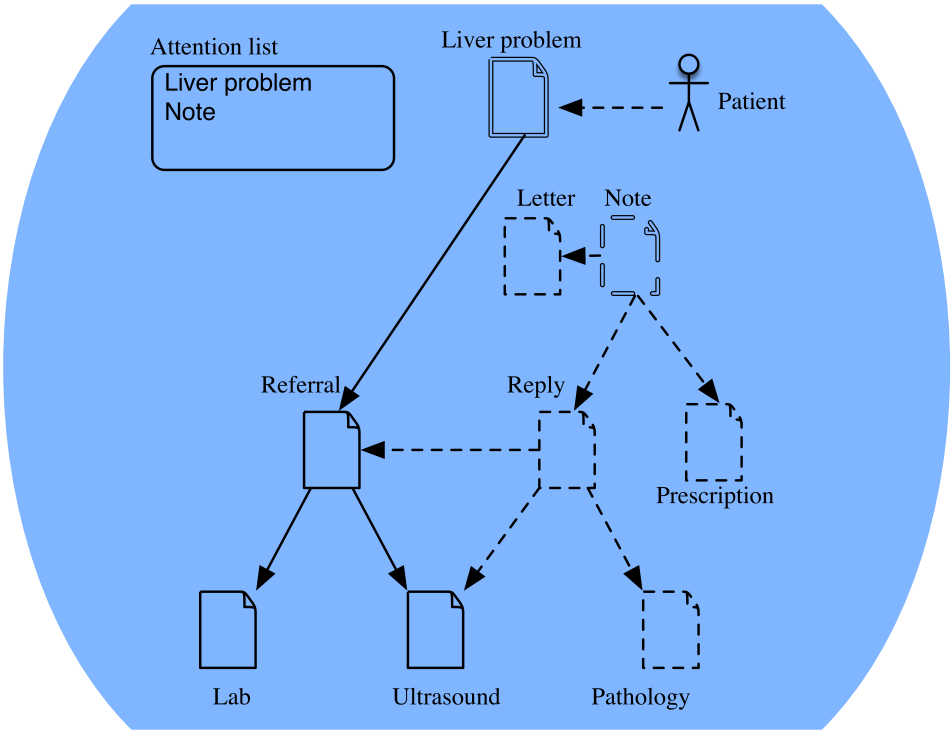


Figure 34.6: The patient sees only handled elements.

If, on the other hand, you want to give the patient explicit access to a not yet handled element in the record, you can do so, which automatically also gives the patient access to the underlying documents that element depends on. See figure 34.7.

Finally, if and when the new note is handled by a doctor and becomes part of the tree rooted in the issue, the patient gets access to that part of the tree as well, automatically. See figure 34.8.

Interestingly, elements at any level which were never part of summaries, or at least not part of any summary the patient or doctor at a higher level ever received, become invisible to this process, which solves another problem, namely how to handle false starts or abandoned hypotheses. Unless these abandoned trails are referred to in any summary that is in the tree we're unravelling, they will become invisible, as they should be.

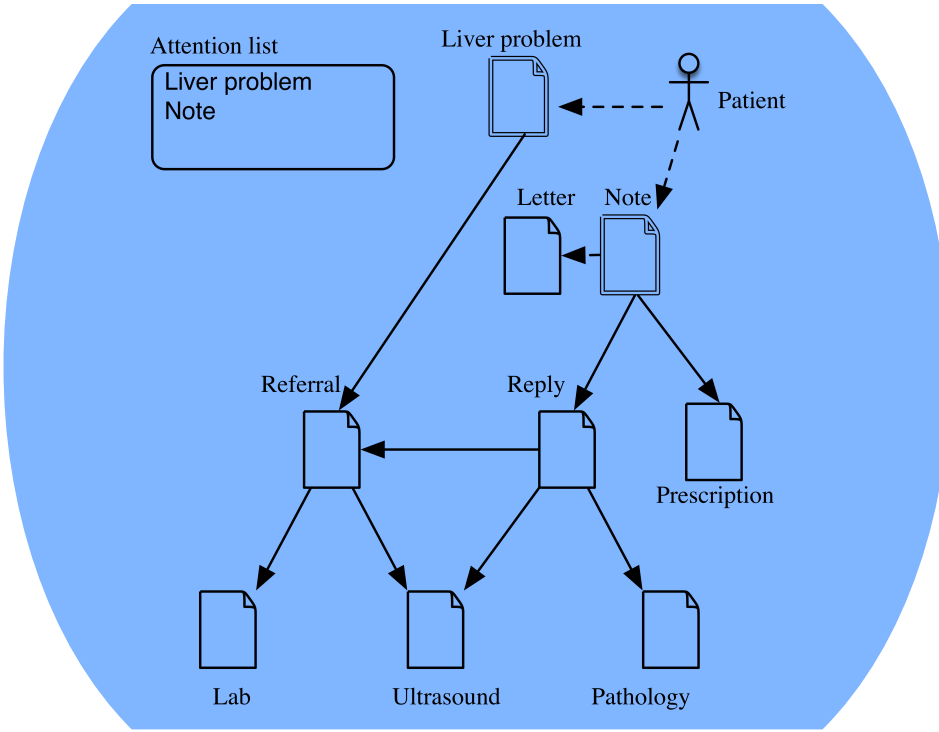


Figure 34.7: The patient can be given direct access to unhanded elements.

This solves several problems, not least of which is how to hide guesswork and scary little side trails from the patient, if the patient has access to the records. If the scary little side trail had no consequences, it won't become part of a trail of reasoning, and thus not a part of the issue tree, so it won't be seen. It's a simple as that.

Another problem we're solving with this pattern is the separation into distinct record systems for different specialities. One way of doing that is hiding all the internal details in a system and only issuing summaries from one speciality to the next. With the document tree principle, we're doing the same thing but on a continuous basis, document by document, summary by summary, making the process much more pervasive, and at the same time with no particular border drawn around specialities. It's a much sounder technical and information theoretical basis to build upon.

But it gets better still. If each summary contains not only the references to the underlying documents, but also the decryption keys (symmetrical) for those documents, the underlying documents could be fully protected

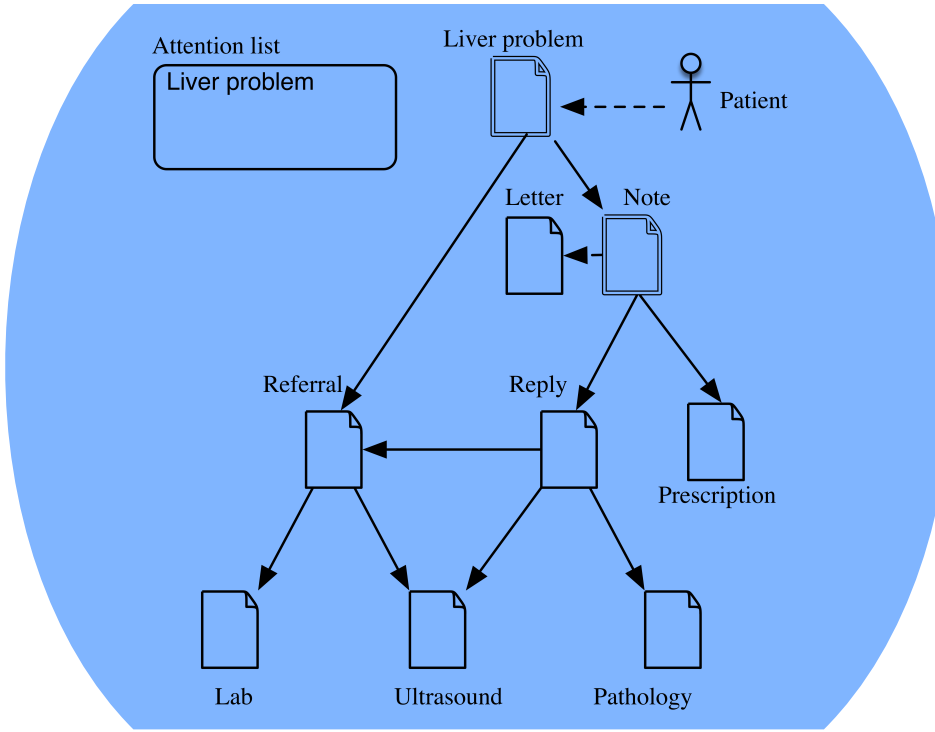


Figure 34.8: When the element becomes handled and linked, the patient also gets access.

from view from anyone who has no reason to see the document. And as "reason to see" the document counts being in the possession of another document that refers to it. That document in turn can only be decrypted if the key is retrieved from yet another document higher in the tree referring to it.

If a particular document is also used in a scientific study of some sort, it will become part of a collective summary for that study, and thus reachable from the study side. The same documents may, but don't have to, be part of some clinical summaries for other purposes. The meaning and function of "primary data source" used in the performance of clinical studies becomes clear and explicit. High five!

It's turtles all the way up, but sooner or later you do reach the last turtle, which could be the patient, the GP, or some kind of database, or a clinical study setup. Each patient could have more than one root, depending on problem areas.

If documents are arranged this way, any constituent documents can have their decryption keys "forgotten" as long as they are part of a higher level summary. Orphaned document keys could be saved in a special database for useless facts, or simply aged out and discarded.

Interestingly, this implies that *all* documents, except the very top level root documents in the tree, can be stored anywhere, even right under the nose of the NSA. As long as the root documents are safe, the other documents can't be decrypted. Root documents themselves can also be set free, as long as the key to each root document is safe. This makes the management of the (momentarily) ultimate keys quite tractable.

The act of "signing off" on received documents, which today is a pretty meaningless hassle, also achieves meaning. As it is today, we look through new results, sometimes signing off on them, but we're never really sure if there is an action presumed on our part, or if the simple viewing of a result has meaning. This confusion is reflected in the implementation of current systems, such that some systems flag results and reports we haven't seen (literally, just like "unread mail" in an email program), while other systems expect us to click a "sign" button, and still others implement both methods in a crazy deluge of contradictory messages. If new results instead are viewed as document references that are not made a part of another higher level summary of some kind, then the action expected of the doctor to remove a result or report from the "new" list, is to incorporate that reference in a summary, which could be a letter to the patient, an outgoing referral, or anything similar. With this system, the actual *action* that makes *actual use* of the result or report, *forms* the signature. There's no ambiguity left. Instead of "unread" versus "read", we have "not handled" versus "handled", or "unused" versus "used", which makes a lot more sense. Every report requires some kind of disposition, or it will stay in the list of the "things that need attention". Neat. (There's a more in depth discussion on these lists of "unused" elements, which I call "attention lists" on page **??**.)

The documents as such, in their encrypted form, can be stored in a distributed hash table form or similar, for maximum accessibility and redundancy, with minimal requirements on local database resilience.

Document checksum

What I'm describing here is the technical design to persist these dependencies, while it's up to the designers to make this process transparent and painless to the user. It could be as simple as a clickable symbol inside a document to open any included references to other documents.

First, we need a unique and reproducible reference to documents in general, and the most obvious choice is a checksum on the canonicalized con-

tents of that document. Yes, that was a mouthful, but it can be explained.

I described what a checksum is, using the SHA function as an example, in the section on digital signatures on page 240. It's simply a sum made from the actual text of the document using a pretty complicated arithmetic process.

A "canonical" version of a document, is a document where we've normalized spaces and other invisible characters, so that differences between systems should not influence the value of the checksum. We can, for instance, stipulate that all trailing spaces in lines should be removed, and we could specify a particular character encoding in the file, and how the end of lines should be marked[18].

Document signature

A document signature is nothing but an encrypted form of the document checksum, where the encryption is done using a public-private key system, with the private key of the person signing the document. Referring back to the section on digital signatures (see page 240), we can summarize this as follows. First we create the checksum (hash) of the document:

$$SHA(M) = M_H$$

Assuming our signer is good ol' Alice, we encrypt this number M_H with Alice's private key $PK_{\text{priv}}(A)$ to get the encrypted version:

$$M_H \overset{PK_{\text{priv}}(A)}{\longrightarrow} M_S$$

The resulting signature M_S can be stored as a pair with the original checksum M_H in a database, allowing us to look up who signed off on a referenced document even without knowing what that document contains. This can be very useful at times. Note that the digital signature M_S contains a reference to the owner of the key, so the identity of the signer can be recovered from the signature.

References summary

We can refer to any element from any other by using the document checksum as an identifier. The document can be stored anywhere convenient,

[18] As an example: some Unix-like platforms like OSX use a single LF character at the end of each line, while Windows and DOS use a pair of characters: CR and LF. Before calculating the checksum we must convert to one of these versions and see to it we always do that, regardless of which machine we're doing the calculation on. If we didn't, a document looking exactly the same to the user would result in two entirely different checksums on OSX and on Windows.

and actually locating the document by this identifier can be done in any number of ways, for instance by using distributed hash tables.

The documents should always be stored in an encrypted form, so to access the contents once the document has been located, you also need a decryption key. Each reference to a document must therefore include two elements: the document reference (which is equal to its checksum), and the decryption key. I'll use symbols as in the following table. The letter "M" denotes "message" in crypto lingo, but in this case the "message" the same thing as "document" or "element of the record".

Table 34.4: Symbols for document references.

Expression	Meaning
M_H	Document checksum (hash), which is also its identifier.
M_K	The symmetric key used to encrypt and decrypt the document.
M_S	The digital signature on the document, created by the originator of the document.
M_R	The reference to a document from another document that is dependent on it.
M_E	The encrypted document.

With this notation, we see that a reference to a document must include two things: the document identifier (so we can locate it), and the symmetric encryption key (so we can read it):

$$M_R = \{M_H, M_K\}$$

The document itself, in its encrypted form, is stored together with its identifier, so it can be located:

$$\{M_H, M_E\}$$

The document signature can be kept separately. It also needs the document identifier so we can find it and link it to the document it signs:

$$\{M_H, M_S\}$$

Since the checksum was calculated on the plain text form of the document, it cannot be verified to be correct until after the decryption of the document. The same limitation applies to the digital signature; the document must be decrypted before it can be verified.