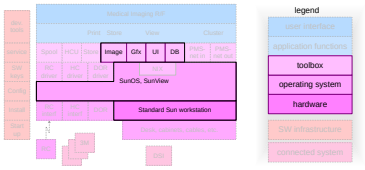
Medical Imaging in Chronological Order

-



Gerrit Muller
Buskerud University College
Frogs vei 41 P.O. Box 235, NO-3603 Kongsberg Norway
gaudisite@gmail.com

Abstract

The chronological events of the product creation of the medical imaging workstation are discussed. The growth in functionality and size from prototype to product is shown. Typical problems in this period are explained.

version: 1.2 status: finished June 23, 2016

1 Project Context

Philips Medical Systems is a very old company, dating back to 1896 when the first X-ray tubes were manufactured. Many imaging modalities have been added to the portfolio later, such as Ultra Sound, Nuclear Medicaid, Computed Tomography and Magnetic Resonance Imaging. Since the late seventies the management was concerned by the growing effort to develop the viewing functionality of these systems. Many attempts have been made to create a shared implementation of the viewing functionality, with failures and partial successes.

In 1987 a new attempt was started by composing a team, that had the charter to create a *Common Viewing* platform to be used in all the modalities. This team had the vision that a well designed set of SW components running on standard workstation hardware would be the solution. In the beginning of 1991 many components had been built. For demonstration purposes a *Basic Application* was developed. The *Basic Application* makes all lower level functionality available via a rather technology-oriented graphical user interface. The case description starts at this moment, when the *Basic Application* is shown to stakeholders within Philips Medical Systems.

2 Introduction

The context of the first release of Medical Imaging is shown in Section 1. The chronological development of the first release of the medical imaging workstation is described in Section 3. Sections 4 and 5 zoom in on two specific problems encountered during this period.

3 Development of Easyvision RF

The new marketing manager of the *Common Viewing* group was impressed by the functionality and performance of the *Basic Application*. He thought that a stand alone product derived from the *Basic Application* would create a business opportunity. The derived product was called Easyvision, the first release of the product was called Easyvision R/F. This first release would serve the URF X-ray market. The *Common Viewing* management team decided to create Easyvision RF in the beginning of 1991.

The enthusiasm of the marketing people for the *Basic Application* was based on the wealth of functionality that was shown. It provided all necessary viewing functions and even more. Figure 1 shows the chronology, and the initial marketing opinion. Marketing also remarked: ”Normally we have to beg for more functionality, but now we have the luxury to throw out functionality”. The addition of

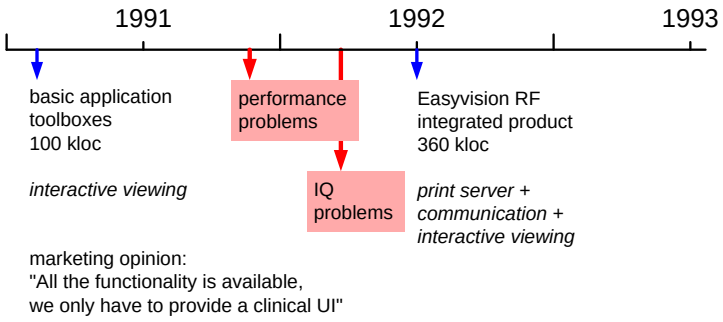


Figure 1: Chronological overview of the development of the first release of the Easyvision

viewing software to the conventional modality products[1] was difficult for many reasons, such as *legacy code and architecture*, and *safety and related testing requirements*. The Easyvision did not suffer from the legacy, and the self sustained product provided a good means to separate the modality concerns from the image handling concerns.

This perception of a nearly finished product, which only needed some user interface tuning and some functionality reduction, proved to be a severe underestimation. The amount of code in the 1991 *Basic Application* was about 100 kloc (kloc = thousand lines of code, including comments and empty lines), while the product contained about 360 kloc.

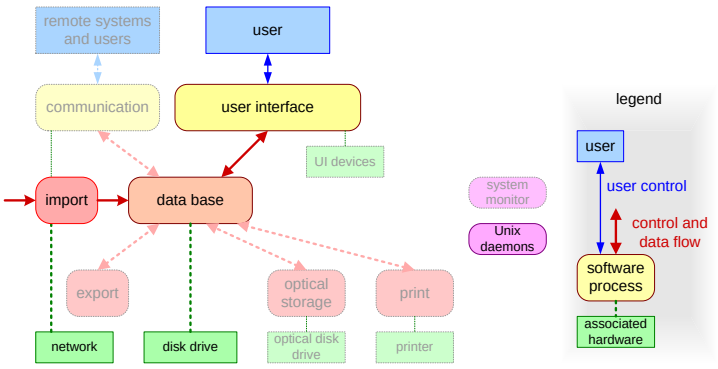


Figure 2: The functionality present in the Basic Application shown in the process decomposition. The light colored processes were added to create the Easyvision

The *Basic Application* provided a lot of viewing functionality, but the Easyvision

[1] *Modality products* are products that use one imaging technique such as Ultra Sound, X-ray or Magnetic Resonance Imaging

as a product required much more functionality. The required additional functionality was needed to fit the product in the clinical context, such as:

- interfacing with modalities, including remote operation from the modality system
- storage on optical discs
- printing on film

Figure 2 shows in the process decomposition what was present and what was missing in the 1991 code. From this process decomposition it is clear that many more systems and devices had to be interfaced. Figures 2 and 3 are explained further in Chapter **??**.

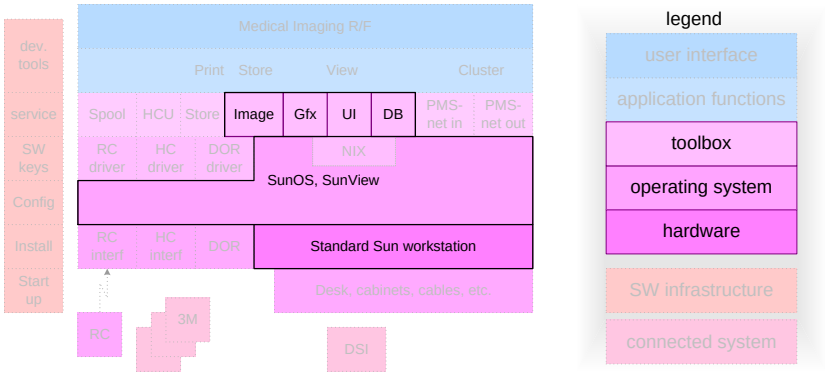


Figure 3: The functionality present in the Basic Application shown in the construction decomposition. The light colored components were added to create the Easyvision

Figure 3 also shows what was present and what was missing in the Basic Application, but now in the construction decomposition. Here it becomes clear that also the application-oriented functionality was missing. The *Basic Application* offered generic viewing functionality, exposing all functionality in a rather technical way to the user. The clinical RF user expects a very specific viewing interaction, that is based on knowledge of the RF application domain.

The project phases from the conception of a new product to the introduction in the market is characterized by many architectural decisions. Architecting methods are valuable means in this period. Characteristic for an immature architecting process is that several crises occur in the integration. As shown in Figure 1 both a performance and a (image quality related) safety crisis happened in that period.

4 Performance Problem

The performance of the system at the end of 1991 was poor, below expectation. One of the causes was the extensive use of memory. Figure 4 shows the performance of the system as a function of the memory used. It is also indicates that a typically loaded system at that moment used about 200 MByte. Systems which use much more memory than the available physical memory decrease significantly in performance due to the paging and swapping to get data from the slow disk to the fast physical memory and vice versa.

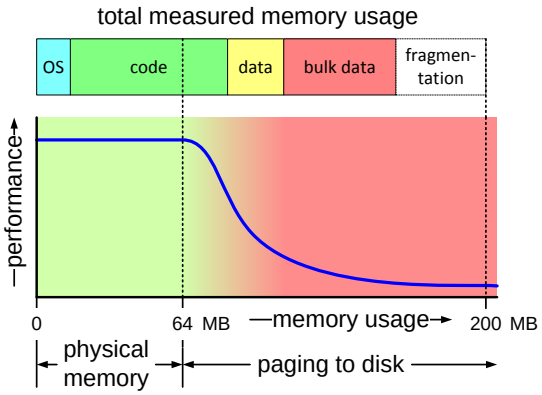


Figure 4: Memory usage half way R1

The analysis of additional measurements resulted in a decomposition of the memory used. The decomposition and the measurements are later used to allocate memory budgets. Figure 5 shows how the problem of poor performance was tackled, which is explained in much more detail in Chapter **??**. The largest gains were obtained by the use of shared libraries, and by implementing an anti-fragmentation strategy for bulk data. Smaller gains were obtained by tuning, and analyzing the specific memory use more critical.

Figure 6 shows the situation per process. Here the shared libraries are shown separate of the processes. The category *other* is the accumulation of a number of small processes. This figure shows that every individual process did fit in the available amount of memory. A typical developer tests one process at a time. The developers did not experience a decreased performance caused by paging, because the system is not paging if only one process is active. At the time of integration, however, the processes are running on the same hardware concurrently and then the performance is suddenly very poor.

Many other causes of performance problems have been found. All of these are shown in the annotated overlay on the software process structure in Figure 7.

Many of the performance problems are related to overhead, for instance for

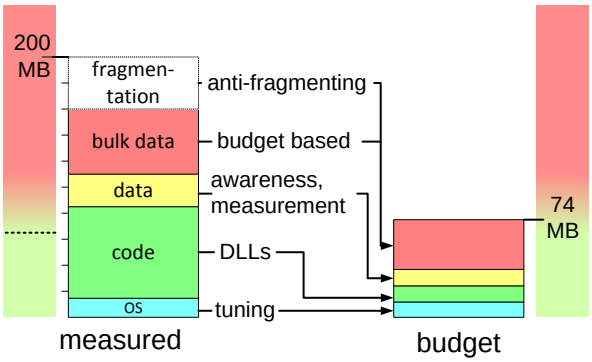


Figure 5: Solution of memory performance problem

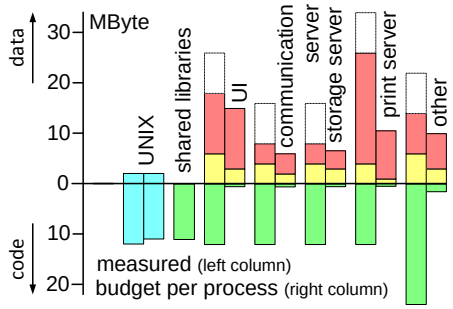


Figure 6: Visualization per process

I/O and communication. A crucial set of design choices is related to granularity: a fine grain design causes a lot of overhead. Another related design choice is the mechanism to be used: high level mechanisms introduce invisible overheads. How aware should an application programmer be of the underlying design choices?

For example, accessing patient information might result in an implicit transaction and query on the database. Building a patient selection screen by repeatedly calling such a function would cause tens to hundreds of transactions. With 25 ms per transaction this would result in seconds of overhead only to obtain the right information. The response becomes even worse if many layers of information have to be retrieved (patient, examination, study, series, image), resulting in even worse response time.

The rendering to the screen poses another set of challenges. The original *Basic Application* was built on Solaris 1, with the SunView windowing system. This system was very performance efficient. The product moved away from SunView, which was declared to be obsolete by the vendor, to the X-windowing system. The application and the windowing are running in separate processes. As a conse-

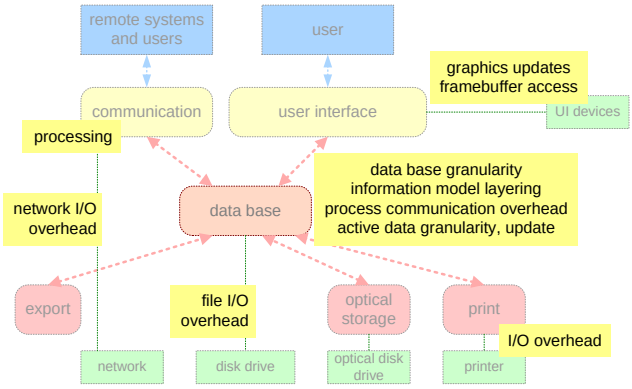


Figure 7: Causes of performance problems, other than memory use

quence all screen updates cause process communication overhead, including several copy operations of screen bitmaps. This problem was solved by implementing an integrated X-compatible screen manager running in the same process as the application, called Nix[2].

Interactive graphics require a fast response. The original brute force method to regenerate always the entire graphics object was too slow. The graphics implementation had to be redesigned, using damage area techniques to obtain the required responsiveness.

5 Safety

The clinical image quality can only be assessed by clinical stakeholders. Clinical stakeholders start to use the system, when the performance, functionality and reliability of the system is at a reasonable level. This reasonable level is achieved after a lot of integration effort has been spent. the consequence is that image quality problems tend to be detected very late in the integration. Most image quality problems are not recognized by the technology-oriented designers. The technical image quality (resolution, brightness, contrast) is usually not the problem.

Figure 8 shows a typical image quality problem that popped up during the integration phase. The pixel value x, corresponding to the amount of X-ray dose received in the detector, has to be transformed into a grey value $f(x)$ that is used to display the image on the screen. Due to discretization of the pixel values to 8 bits *false contours* become visible. For the human eye an artefact is visible between pixels that are mapped on a single grey value and neighboring pixels that are mapped on the next higher grey value. It is the levelling effect caused by the discretization that becomes visible as false contour. This artefact is invisible if

[2] A Dutch play on words: *niks* means nothing

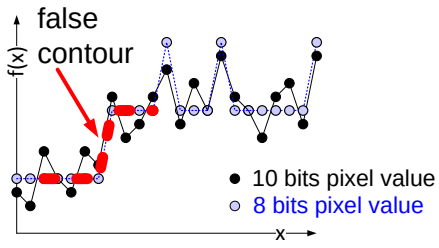


Figure 8: Image quality and safety problem: discretization of pixel values causes false contouring

the natural noise is still present. Concatenation of multiple processing steps can strongly increase this type of artifacts.

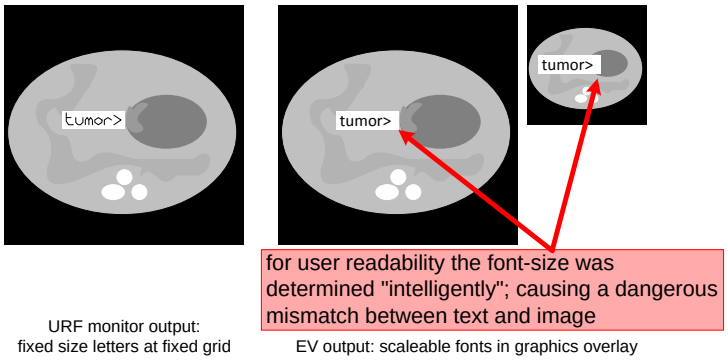


Figure 9: Safety problem caused by different text rendering mechanisms in the original system and in Easyvision

The original design of the viewing toolboxes provided scaling options for textual annotations, with the idea that the readability can be guaranteed for different viewport sizes. A viewport is a part of the screen, where an image and related information are shown. This implementation of the annotations on the X-ray system, however, conflicts in a dangerous way with this model of scalable annotations, see Figure 9.

The annotations in the X-ray room are made on a fixed character grid. Sometimes the '>' and '<' characters are used as arrows, in the figure they point to the tumor. The text rendering in the medical imaging workstation is not based on a fixed character grid; often the texts will be rendered in variable-width characters. The combination of interface and variable-width characters is already quite difficult. The font scaling destroys the remaining part of the text-image relationship, with the immediate danger that the annotation is pointing to the wrong position.

The solution that has been chosen is to define an encompassing rectangle at the

interface level and to render the text in a best fit effort within this encompassing rectangle. This strategy maintains the image-text relationship.

6 Summary

The development of the Easyvision RF started in 1991, with the perception that most of the software was available. During the developement phase it became clear that a significant amount of functionality had to be added in the area of printing. Chapter **??** will show the importance of the printing fumctionality. Performance and safety problems popped up during the integration phase. Chapter **??** will show the design to cope with these problems.

References

[1] Gerrit Muller. The system architecture homepage. `http://www.gaudisite.nl/index.html`, 1999.

History

Version: 1.2, date: March 16, 2004, 2003 changed by: Gerrit Muller

- added short explanation the term ”modality products”
- changed status to finished

Version: 1.1, date: February 27 2004, 2003 changed by: Gerrit Muller

- added section ”Summary”
- added short explanation of the figure memory use per process.
- changed status to concept

Version: 1.0, date: October 30, 2003 changed by: Gerrit Muller

- added section Project context
- fine-tuned text
- changed status to draft

Version: 0.1, date: September 29, 2003 changed by: Gerrit Muller

- fine-tuned text

Version: 0, date: May 21, 2003 changed by: Gerrit Muller

- Created by refactoring ”Threads of reasoning in the medical imaging case”