Execution Architecture Soft Real Time design

-

logo
TBD

Gerrit Muller
University of South-Eastern Norway-NISE
Hasbergsvei 36 P.O. Box 235, NO-3603 Kongsberg Norway
gaudisite@gmail.com

Abstract

version: 0.2 status: planned September 1, 2020

1 Introduction

The soft real time behavior of first releases of products is often disappointing. For instance the user interface feels sticky, due to poor repsonse times, or the system throughput is less than expected.

The soft real time design is competing for architect attention with many other design concerns, such as functionality, quality of service, maintainability et cetera and with project management concerns, such as outsourcing, purchasing, work-breakdown and work-allocation, see figure 1

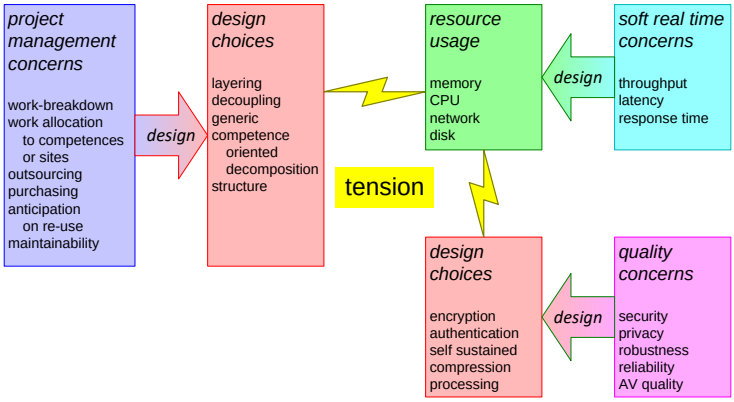

Figure 1: The tension between project management oriented concerns and soft real time concerns

Many of these concerns not only compete for architect attention, but create also design trade-offs: layering and decomposition eases many project management issues, but the additional interfaces and run time interface transitions might kill the system throughput.

Many of the design decisions around soft real time performance are not strictly execution architecture decisions, although sometimes heavily intertwined. For instance using separate processes for functionality developed on different sites heavily impacts the execution architecture.

2 Bloating

The performance of soft real time systems is for a considerable part determined by the amount of bloating of the software

Figure 2 show an actual example of part of the Medical Imaging system [2], which used a platform based reuse strategy. The first implementation of a ”Tool” class was overgeneric. It contained lots of *if-then-else*, *configuration options*, *stubs*

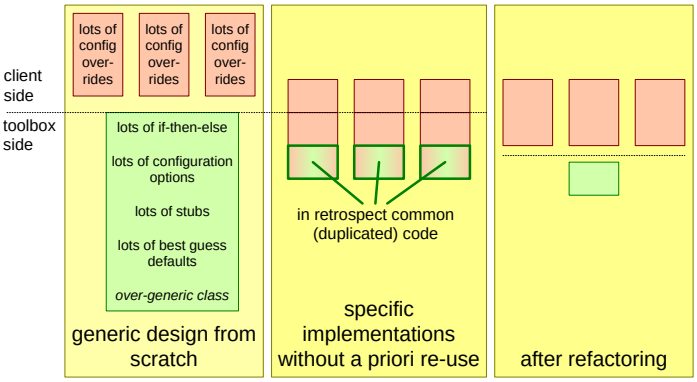

Figure 2: The danger of being generic: bloating

for application specific extensions, and lots of *best guess defaults*. As a consequence the client code based on this generic class contained lots of *configuration settins* and *overrides of predefined functions*.

The programmers were challenged to write the same functionality specific, which resulted in significantly less code. In the 3 specific instances of this functionality the shared functionality became visible. This shared functionality was factored out, decreasing maintenance and supporting new applications.

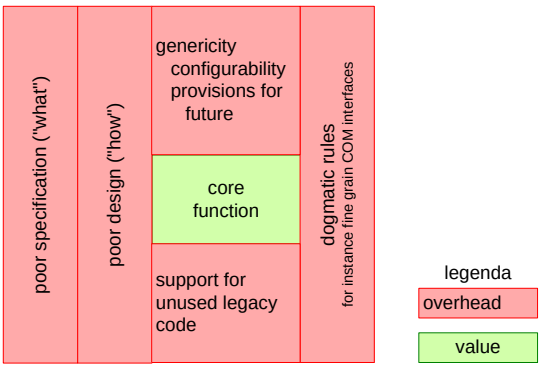

Figure 3: Exploring bloating

Bloating is one of the main causes of the *software crisis*. Bloating is the unnecessary growth of code. The really needed amount of code to solve a problem is often an order of magnitude less than the actual solution is using. Figure 3 shows a number of causes for bloating.

One of the bloating problems is that bloating causes more bloating, as shown

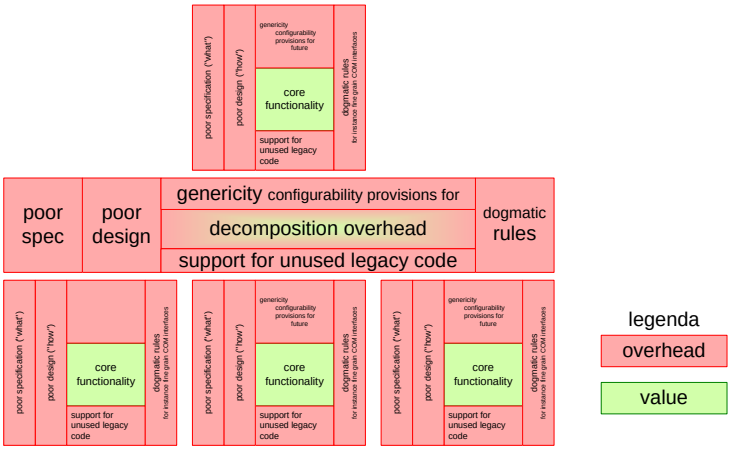

Figure 4: Bloating causes more bloating

in figure 4. Software engineering principles force us to decompose large modules in smaller modules. ”Good” modules are somewhere between 100 and 1000 lines of code. So where unbloated functionality fits in one module, the bloated version is too large and needs to be decomposed in smaller modules. This decomposition adds some interfacing overhead. Unfortunately the same causes of overhead also apply to this decomposition overhead, which means again additional code.

All this additional code does not only cost additional development, test and maintenance effort, it also has run time costs: CPU and memory usage. In other words the system performance degrades, in some cases also with an order of magnitude. When the resulting system performance is unacceptable then repair actions are needed. The most common repair actions involve the creation of even more code: memory pools, caches, and shortcuts for critical functions.

The overall aspects of bloating are devastating: increased development, test and maintenance costs, degraded performance, increased harware costs, loss of overview, et cetera.

References

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

[2] Gerrit Muller. Case study: Medical imaging; from toolbox to product to platform. `http://www.gaudisite.nl/MedicalImagingPaper.pdf`, 2000.

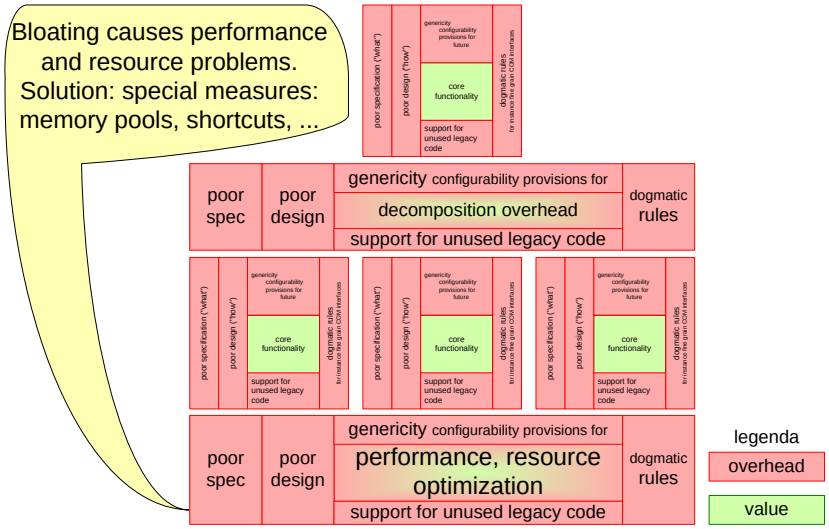

Figure 5: causes even more bloating...

History

Version: 0.2, date: March 7, 2003 changed by: Gerrit Muller

- added bloating text

Version: 0.1, date: December 6, 2002 changed by: Gerrit Muller

- added overhead penalty of modularity and function call
- added bloating figures

Version: 0, date: October 30, 2002 changed by: Gerrit Muller

- Created, no changelog yet