

Measuring Architecting Effort

Eelco Rommes¹, André Postma², Pierre America¹

¹Philips Research, Eindhoven, The Netherlands; ²Philips Medical Systems, Best, The Netherlands
{Eelco.Rommes, Andre.Postma, Pierre.America}@Philips.com

Abstract

The amount of architecting effort is a factor in a software development project's efficiency. Before steps can be taken to optimize this factor, its current position must be known. We have measured the amount of architecting done in two industrial cases relating to the development of medical imaging systems. We discuss these cases and some of the problems that we encountered.

1. Introduction

Architecting has an important role to play in the efficient development of complex, software-intensive systems [1,5]. Recently, Boehm and Turner have shown the relationship between the architecting effort in a project and the amount of rework that has to be carried out later [2].

Their results show that if not enough attention is devoted to architecting, this will result in a considerable amount of rework later in the project - up to 91% added time for large systems [2]. On the other hand, devoting too much effort to it is a waste of resources. This means that optimization of the architecting effort can make a development process more efficient.

The optimum amount of architecting effort for a project, known as the architecting *sweet spot*, is largely dependent on the size of the software under development. As a general rule, the larger the system is the more architecting it needs. This can range from 5% of the total project time for 10 KSLOC systems up to almost 40% for software with 10,000 KSLOC [2].

In order to see if improvements can be made, an organization needs to know how much effort it is currently devoting to architecting. We have measured the architecting effort in two case studies around the development of medical imaging systems.

2. Measuring Architecting Effort

Philips Medical Systems [4] develops a wide range of medical imaging systems for the creation and storage of medical images to facilitate diagnosis, treatment planning and interventions to cure various diseases.

The departments that develop these systems generally evolve an existing code base, with a new system release

approximately once a year. Whenever possible an effort is made not to have to create an entirely new architecture because this involves an enormous amount of work and risk, such as delayed system delivery and extra development and testing costs.

We examined projects in three development departments at Philips Medical Systems, with the intention of finding out their position relative to their architecting sweet spot.

2.1 Case study: Daisy

The Daisy department is responsible for a line of medical imaging systems. The code base for these systems contains five million lines of code and approximately the same amount of test code.

In this case, the department was able to supply us with quantified data on past projects, because its development process included a timekeeping task for the entire development staff. The Daisy timekeeping database included ten categories of which we identified four to be architecting tasks: requirements management, analysis, design, and risk management.

A fifth category, 'general', belonged only partly to architecting. This was due to the fact that a Daisy timekeeping database is not created right at the start of a project. Instead, a period of time is allowed to elapse during which timekeeping is carried out without any specification of categories. All effort booked during the start-up phase of a new project ends up in the 'general' category of the timekeeping records. Even after the other categories become available people do still book hours in the 'general' category and the final data in the 'general' category is thus made up of truly 'general' effort as well as all the effort that was registered before the more specific categories became available. Typically, a lot of the effort that is invested at the start of a project is up-front architecting work, and all this data gets buried in the general category.

In the end, we were able to calculate a lower and upper bound for Daisy's architecting effort, with a maximum error of 9%.

2.2 Case study: Daffodil

The Daffodil department develops and maintains another line of imaging systems. Daffodil does not have a time-tracking process like Daisy does, and we therefore

needed to take a different approach to data collection. Here, we took information from interviews and documents instead.

We chose a representative set of people who performed architecting work and asked them to estimate the amount of time they spent on architecting.

We had to take into account that there are many different architecting roles within Philips. Partly, this is a reflection of the complexity of the systems developed. To handle this complexity, the design is divided in subsystems at many levels of abstraction. At each level, a different design role can be identified. Thus, there are system designers, software architects, system architects, principal architects, senior architects, and fellow architects, all of whom do different jobs involving architecting in one form or another.

To deal with this variety, we gathered estimates of the percentage of time spent on architecting for each of these roles. Combined with data on the number of full-time equivalents (fte's) per role, we were able to obtain an estimate of the amount of architecting effort spent.

Interestingly enough, it turned out that roles only influence marginally the amount of architecting that a person is able to perform. Roles do influence very much what persons spend the rest of their time doing, however. For example, a fellow architect may attend strategy meetings, whereas a system designer may spend part of his time coding.

Using the estimates we gathered, we were able to calculate an estimate of the architecting effort at Daffodil.

3. The Sweet Spot May Be Sour

When interpreting architecting effort measurements, it should be noted that there might be valid reasons why an organization is not on its sweet spot.

Boehm and Turner mention the influence of requirements volatility: projects with highly volatile requirements benefit from performing less up-front architecting in order to prevent documentation rework. In other words, their sweet spot moves to the left.

Also, the assumption that an organization seeks optimum efficiency can be false. A process may be optimized on the basis of other aspects. For example, product quality may be crucial, with the result that more time is spent on getting the architecture right at the expense of development efficiency. Alternatively, product quality may be sacrificed in favour of speed in order to get a product out to the market quickly. In such cases an organization should expect to be situated right or left of its theoretical sweet spot.

Several of the people with whom we worked to evaluate the method said that their department benefited from work done in an earlier project with a strong focus on architecture. It seems that careful architecting can keep the architecting sweet spot for subsequent projects to the

left. On the other hand, it is to be expected that a lack of focus on architecture will force the sweet spot to the right: effort is needed to refactor the code and make it possible to develop the next system release. This effect is also known as *technical debt* [3].

Being on the sweet spot does not mean that the architecting process cannot be improved. The method described here only measures the amount of architecting effort performed; it says nothing about the quality of this effort. Even if enough architecting has been carried out, this architecting may well have been carried out in the wrong way.

4. Conclusions

We have discussed our experiences in measuring the architecting effort in two industrial case studies. Although it is relatively easy to obtain a reasonable number, interpreting the meaning of the measurement can be tricky, as many factors play a role that influence the optimum amount of architecting at a given development organisation. Exactly what these factors are, is a topic for further research.

References

- [1] L. Bass, P. Clements, and R. Kazman: *Software Architecture in Practice*. Addison-Wesley, 1998.
- [2] Barry Boehm and Richard Turner: *Appendix E2 - How Much Architecting Is Enough? A COCOMO II Analysis*. In: *Balancing Agility and Discipline: A Guide for the Perplexed*, Addison Wesley, August 2004.
- [3] Wiki Wiki Community: *TechnicalDebt*. <http://www.c2.com/cgi/wiki?TechnicalDebt>, December 2003.
- [4] Philips Medical Systems: *Philips Medical Systems website*. WWW, May 2005. <http://www.medical.philips.com/main/>
- [5] E. Reichtin and M. W. Maier: *The Art of Systems Architecting*. CRC Press, Boca Raton, 1997.