Final Report

PMoC
4/14/2003
Version 1.0

T-76.115 Project Plan
PMoC

Version	Date	Author	Description
1.0	14.4.2003	JF	“First” version

Table of Contents

1. Introduction

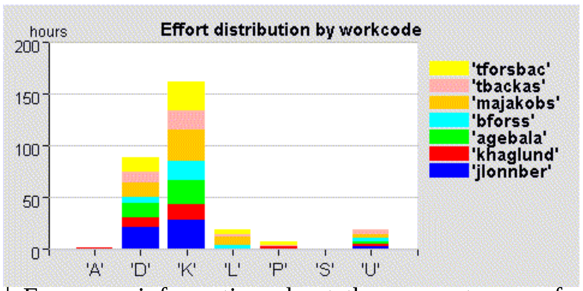
During the last three to four years, the customer (VTT) has been developing a new concept for making configurations and storing and running process simulation models. Old software exists, that has been used during a number of years, but this software has no standardized interfaces for exchange of information with other software. The new concept developed by VTT uses GML and SVG formats, both subsets of XML, making it possible on a much wider range to have good intercommunication and data exchange all over the world in the context of process simulation. Also the old software had several limitations due to it's isolation as a stand-alone software.
The main objectives for the PMoC (Process Model Configuration) project, carried out as a course in software production (T-76.115) at Helsinki University of Technology was to create a software that could verify the newly developed concept at VTT, and to demonstrate new possibilities of the concept[1].

2. Project progress

2.1 Project Planning phase

As we started of in September our group had a huge amount of information to assimilate, as both the newly developed GML format, and the SVG graphics format was generally new for all of us. No one had any thorough experience of project management and organization either, so on that area we were more or less forced to try our steps forward.
Despite that most of the group members barely knew each other from before the cooperation was very good from the beginning. Several 'lessons' were held by our customer on the new concept for us to understand what to do.
Documentation was being created based on our customer meetings, and the foundation for the program architecture was made by our designers. Also the group was divided into logical subgroups for different parts of the project, and responsibility areas were defined.
Additionally several software packages for SVG and programming IDEs were tested for suitability for this particular project.
The project seemed very interesting, even though there was a growing understanding within our group that we still had enormously much to learn to carry through the project, and this feeling remained to the end of the fourth phase of the project.

The biggest problems during this phase were to have a grip what everyone should do, and which parts of planning and documenting to emphasize on. Still at the end of the phase when the delivery was made, we shared a strong feeling of good cooperation within the group.



As can be seen from the graph to the right, the most time consuming during the first phase

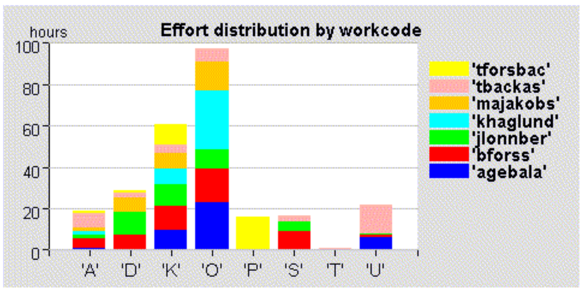
[1] For more information about the concept, we refer to the doctoral thesis by Tommi Karhela "A Software Architecture for Configuration and Usage of Process Simulation Models" published by VTT Technical Research Centre of Finland 2002

were the meetings, secondly also writing the documentation was a major part of this phase.

2.2 First implementation phase

The phase contained several more parts than the last one, and consisted mainly of planning and organizing the project, designing the architecture , having a few meetings with the customer, planning and doing the basic parts of the implementation and writing and updating the required documentation.
As for a "normal" project in the context of this course, most information required to produce the software would have been assimilated already, but for our group this had to be an ongoing process through the whole project. Thus, at the end of this phase we felt that now we could start our first implementation phase. As one of the group members commented, there was always the feeling of laying on phase behind.

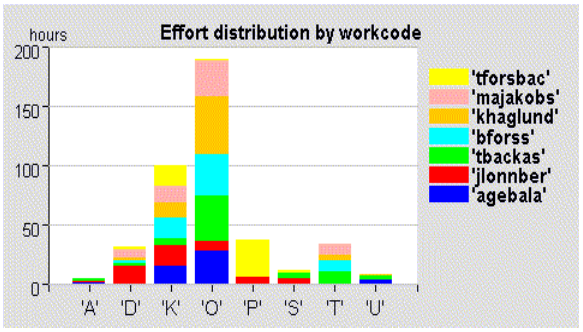
Already at this stage it was to some extent realized that the needs of communication was different from the first phase, and efforts were put to correct this problem to the second implementation phase.
During this first implementation phase we tested having a day to work together all in one classroom to get more done at a time, and to improve the communication during coding. It was a good experience, and this was applied several more times during the latter phases.
Also a mistake had been made when planning the schedule for this phase, so the group had one week less to finish this phase, but actually that mistake worked as a highly motivating factor to get things done at time.



Programming got well of, but still many meetings had to be held.

2.3 Second implementation phase

The second implementation phase was basically a good experience, and went over all quite well. This phase had main emphasizes on programming, still much meetings needed to be held both internally with the group, and with the customer. We tried to minimize such meetings were everyone should be present, and focused on holding meetings with only needed members for different parts of the project. Also customer meetings were held in smaller groups.
Because of this a new problem developed, all though documents were updated based on the meetings, especially the customer meetings, the documents were not a good enough channel to communicate the architecture and features to be implemented.
So therefore the project manager developed what we call "weekly todos" that clearly state what each subgroup and member should have done by a specific date. Even though these documents weren't published weekly they filled their purpose very well. Often they specified more work than was done during a week, so they mostly served for 2 – 3 weeks forward as guidelines for the implementation and other things to be done on a practical level.



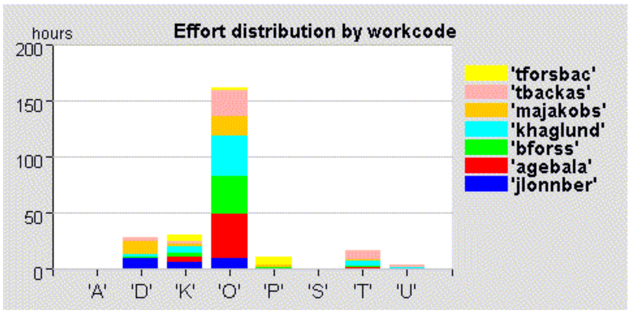
Programming advanced well during this phase, and meetings could be kept on the same level as last phase, that is ~24% of total hours spent on this phase

2.4 Third implementation phase

Finally, in the third implementation phase programming got the very main focus, and even the meetings could be put to a minimal level. (See figure below). The design and documentation also required minimal efforts, thus giving room for good progress in the implementation. Most parts of the software were on a decent level enough for the customer to be able to start using it to verify the GML, and to use the software as it was intended. The user interface had evolved enough and the functionality developed the such an extent that we could see what the program would be like.

Peer testing was done, but the results from our peer testing group, regarding our own software wasn't *that* much of help to us, but this could be expected, and only proved the high degree of complexity in this software.

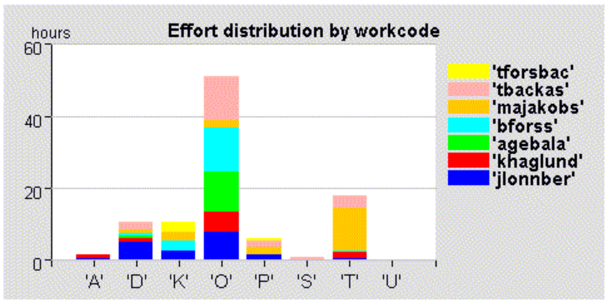
Although a growing need for coordination was expected, the project manager was happy to establish that the subgroups were able to work quite independently, and that most coordination could be done on a lower level. This positive development continued during the last phase of the project.



Project management could be cut down to half of that of last phase (to 4.4% of total phase efforts) and programming could take 62.2%.

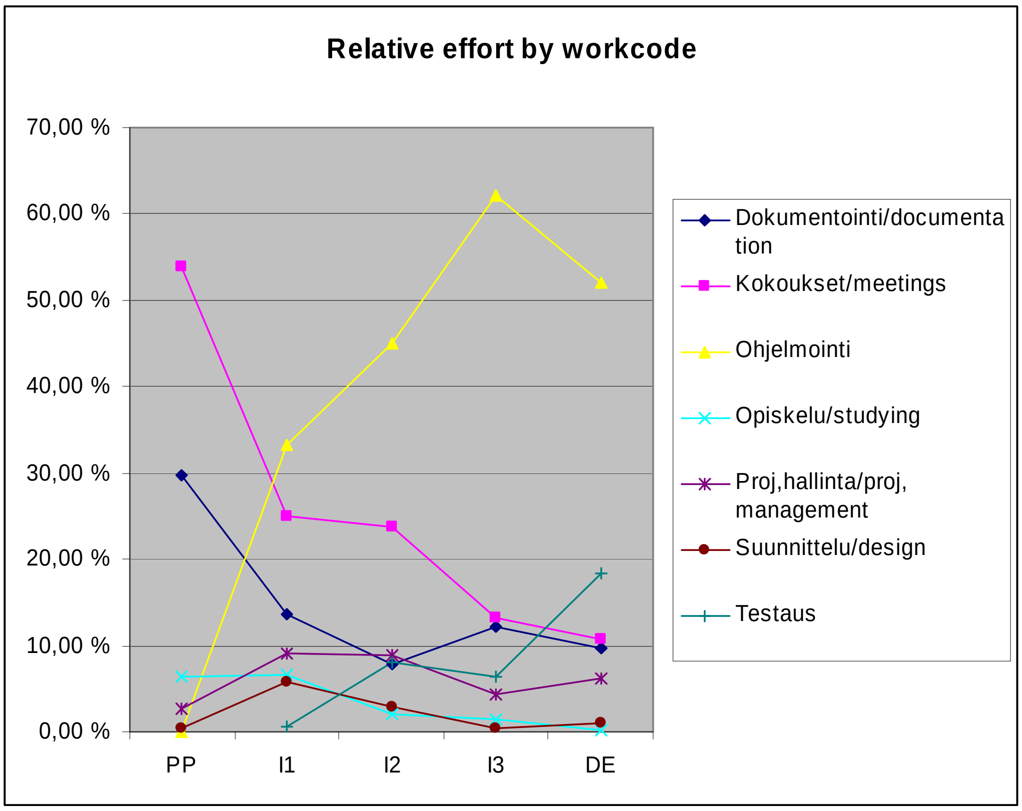
2.5 Delivery phase

A hectic race to finish up the software started, mostly thank to the customer who needed a functioning version for some reasons. This motivated the entire group very well to get all parts working well, and also parts that had been most ‘behind’ finally caught up.
What also has characterized the tree last phases is that limiting the project scope together with the customer has worked very well. There was growing anxiety about scoping, both that we couldn’t scope the project well enough, or that too much scoping would degrade the software not to feel complete enough, but eventually this was done quite successfully.



Programming 52% and project management, documentation and meetings together 27% can be considered quite good.

2.6 Overview



From this diagram the different work areas can be seen, relatively to the total work effort for each phase.

3. Results

3.1 Reaching objectives of the customer

The objectives of the customer were divided in two different parts:

1. **Verification of GML**
   1.1. Verification of the graphical model on two hierarchical levels including references to the topological model

- Implemented with *unlimited* number of hierarchical levels.

   1.2. Verification of the graphical and topological model on N hierarchy levels, with only static lifting of terminals.
   1.3. Verification of the graphical and topological model on N hierarchy levels, also with dynamic lifting of terminals.

- Static and dynamic lifting of the terminals were moved into the future when scoping the project, and were not considered vital for the customer to be able to verify the GML concept

2. **Post delivery development**
   2.1. Internal interfaces should be designed so that the query forms of the components can be developed later on.
   2.2. Internal interfaces should be designed to ease later adding of trend- and monitor data retrieving.

- Design took this into account, so query forms and trend- and monitor data retrieving should be relatively easy to add.

   2.3. Internal implementation logics and data structure should be designed to ease integration with the Galleria architecture later on.

- Fully taken into account, everything was designed (and redesigned where needed) to integrate with Galleria architecture.

   2.4. Rotation, flipping and snapping-to-grid of symbols should be easy to add later on.

- Implemented.

   2.5. Animation properties should be possible to add later on.

- Was not considered very important at this stage.

Project plan: "*The goals of the project are to create a system that follows as many of the customers wishes as possible, but concentrating on the most essential goals of the customer, being the verification of the GML -specification; either showing that it is a working design or pointing at the problems of the design - possibly also correcting possible such problems. The* *most central features of the system, being xml-handling and limited editing of both components and the actual connecting of components into a whole, are to be created. Any of the many possible additional features are a plus, but not necessary for the project outcome.*"

As verification of the GML very well can be made with the software, and even additional post-delivery functionality has been implemented, the project should be considered a success to most part. As can be seen from the comments above about the different goals, some of them were removed when limiting the scope of the project. But on the most important parts the objectives were fell fulfilled.

3.2 Group objectives

From project plan: "*… By doing this, the project group also especially expects and wishes to learn about the whole project process, and not only the possible extra lesson in programming practice, although the programming environment is new to the whole group. The project groups also wants to finish the project within the time set for the task, and therefore limit the project to a reasonable level of complexity.*"

The goals were very high in the beginning, and revised several times, still the software can be considered quite complex. The general feeling in the group is that despite many though tasks and challenges we have done quite well and that the result is very good, considering the level of difficulty of the project. Additionally the technical specification contains a lot of useful functionality that has not yet been implemented, which is of good value for the customer for further development.
Definitely all the group members have also learned a lot about team work, and about making a bigger software as a team. The efforts put to keep within the time limit have also shown fruitful, as the total project time hasn't even reached 1500 hours yet. So the time budget has only been exceeded with about 100 hours, which should be considered very low compared to the complexity of the project.

3.3 Inspection group and customer feedback

As already stated, the inspection group didn't get much hold of the software, even with a thorough and good user manual. That should testify by itself of the complexity of the program. without a user manual, a possibly bad user interface could be blamed, but not in this case. Generally at that stage the inspection group considered the software very half-done, which also was the case.

The customer has generally been satisfied with the progress of the software, and also with the reached final result. At times the customer has given constructive critics about the software, regarding both architecture and usage of the software, which has been greatly helpful.

4. Work methods and tools

4.1 Process

Our original plan was to follow the course's USDP/RUP process guidelines as closely as possible. We had to follow these guidelines quite closely anyway, as the course requirements included several documents specified by the process. In practice, we deviated quite a lot from the process.

Most of the project planning phase and the beginning of the I1 phase were spent trying to understand what the customer wanted and planning the methods to use. During the implementation phases, we went into cycles containing meetings with the customer, updates to requirements and specification documents, implementation and evaluation by the customer. The length of these cycles varied a lot, but we had one or two of these every phase, with one of the cycles coinciding with the end of the phase. Also, at the end of each phase, we produced the documents required for that phase's delivery. We performed organised testing at the ends of the second and third implementation phase and during the delivery phase, and relied on adhoc and unit testing during implementation. As the customer wants to develop the program further, deployment simply involves FTPing the code (wrapped in a package) to the customer.

All in all, our slightly adapted process worked pretty well, although implementation never proceeded as fast as we would have liked it to.

4.2 Methods

Aside from the process specified by the course, we added some other methods. One of us was responsible for each method.

We found that using CVS solved most of our version management problems for code, and using the group's discussion forum was adequate. We will probably use CVS for future projects, but the discussion forum was not ideal.

Our communication and meeting practices were satisfactory, but there was room for improvement. Communication and meetings are pretty much unavoidable and important. Toward the end of the project, we managed to decrease the amount of meetings noticeably, which was good as we were running over budget on time.

Although we got off to a good start initially with coding conventions, toward the end the various developers' own styles started to show. This was compunded by lack of time, different and partially incompatible editors (especially as regards indentation). The result is that a lot of the code is badly commented, not documented according to Javadoc specs and inconsistent in form. It seems that it would be useful to pay more attention in the future to coding conventions.

The nightly builds showed that we never messed up the code in the CVS so badly that it couldn't compile, but we failed to integrate tests into the nightly builds sufficiently well for them to be useful. With better automated testing, these nightly builds could be useful in the future.

The use cases were used to write the original technical specification, but after that they were mostly ignored. The use case diagrams were considered especially useless. As a way to produce initial specs, we believe use cases can be useful in similar projects.

Our modified V-model testing proved to be quite useful, although testing started a bit late due to delays in implementation. In the end, the tests comprised automated unit tests (built by the developers for their own code), some ad-hoc integration tests and system tests based on the use cases. The V model fits our testing needs quite well, as it allows us to find most bugs quickly while ensuring that the system works as a whole, and as such it would seem to make sense in the future.

For further details, see the separate documents for each method.

4.3 Tools

We initially tried to use Visual C++ for development, but quickly changed to Sun's JDK, as we were having problems finding suitable SVG libraries for C++ (and hated Visual C++'s GUI libraries). As we found an IDE desirable, we switched to AnyJ, and later partially to JBuilder. This inconsistency in IDE caused some cosmetic problems with the code, but did not impair functionality. In the future, we will probably use some IDE for ease of use, but this will depend of the language used.

Mostly we used CVS and a web-based discussion forum (Phorum) for version management and communication, although we initially shared documents using a file server at SoberIT, which was found to be unsatisfactory due to lack of remote access. CVS was found to be quite useful.

Time reporting was done using Tirana, bug and size reporting using Burana, code size counting using CCCC (as suggested by the course staff) and nightly builds using ANT. ANT could be useful in the future.

5. Aftermath

Generally the software has evolved according the plans. The work amount at the end is actually perhaps even lower than what was expected as a worst case. (1500 hours, as stated above). Much was implemented, but still there is much functionality that could be added, eg.

- Component type editing
- Tabbed tree view
- Symbol toolbar
- cut / copy / paste / undo
- support for several departments / servers / hosts
- memory usage optimization
- better select tools for the canvas

6. Educational value

6.1 Member comments

Member 1

During this course I have learnt the basics of "a good" software project in practice; the importance of meetings, documentation, testing, design, coding and communication. Although this hasn't been a full time job, it has worked out pretty well, and you have gotten the feeling of a real project. The personal method (coding convention) has taught me about the possibilities of software engineering methods and the gain you can get out of them.

Member 2

No matter how realistic you try to be, all your plans are too optimistic by far.
Nobody reads specifications, and if they do, they can't understand them.
Debugging other people's code is a great way to waste time.
Never tell people the _real_ deadline if you want things on time.

Member 3

It’s important to spread to workload evently, and that the persons get along with each other in the group. To be honest though, I have personally learned most about Java as a programming language, and not any specific project related method as probably was the point.

Member 4

I learned that a clean and modular stil is best when it comes to both program architecture and low level implementation. During a project with a long lifetime, this reduces troubles efficiently. A big group needs good communication and work methods and document should be short in order for the group to use the and keep them up to date. I also found that keeping good momentum and tempo in the work is important when it last a long time.

Member 5

Technically, I learned a lot about XML and Batik. I have also learned the value of good cooperation, without it a project cannot succeed, and that the customer needs to be closely involved in the project for it to succeed.

Member 6

I learned a lot about SVG and DOM during the course. Even though they were familiar to me previously, I have now gained a considerable amount of experience in using them. Using the Batik library was also learning experience.

As the test manager I learned about planning and coordinating tests and got more familiar with V-model and other testing methods and concepts. Doing a big project from scratch all the way to the final product was a nice experience.

7. Course feedback

The most prominent thing about this course is that its really hard work. Many of the group members think the course should at least be 6 credits, this way there would be more time for software methods and tools. The mentoring on the course was excellent and helped us a lot. Our customer was also very helpful and put some real effort in the project. The documentation was not to the whole groups liking, especially in the beginning. Burana was quite awful to use, and Tirana was not fast either, although it served its purpose. As an improvement suggestion, he course could provide a Java IDE or two, e.g. JBuilder or AnyJ, on HUT-CC or TA218 machines as many groups use Java in their project.