V-model testing

PMoC
14/04/2003
Version 1.5

Version	Date	Author	Description
1.5	14.04.2003	Jakobsson	Added experiences for DE and an evaluation chapter
1.4	20.03.2003	Jakobsson	Added experiences for I3
1.3	07.02.2003	Jakobsson	Added experiences for I2
1.2	2.12.2002	Jakobsson	Added experiences for I1
1.1	18.11.2002	Jakobsson	Revision
1.0	28.10.2002	Jakobsson	First version

1. Introduction

The V-model was originally developed from the waterfall software process model. The four main process phases – *requirements*, *specification*, *design* and *implementation* – have a corresponding *verification* and *validation testing* phase. Implementation of modules is tested by *unit testing*, system design is tested by *integration testing*, system specifications are tested by *system testing* and finally *acceptance testing* verifies the requirements. The V-model gets its name from the timing of the phases. Starting from the requirements, the system is developed one phase at a time until the lowest phase, the implementation phase, is finished. At this stage testing begins, starting from unit testing and moving up one test level at a time until the acceptance testing phase is completed. [1]

The V-model is easy to understand, but the timing of the phases where testing occurs after all implementation is done, is not an adequate approach in today's iterative software processes. However, the test levels from the V-model which associate a certain process phase with a testing phase is still relevant and will be used in this project. When the test levels from the V-model are used in parallel with development, and more importantly, when the different levels are tested simultaneously in multiple iterations, the V-model testing method can become an effective and manageable testing process.

2. The use of V-model testing in the ProConf project

Testing will be done according to the V-model, but adapted to the iterative nature of the software development process that is used. During development the program will be tested at all levels simultaneously. The different levels are unit tests, integration tests, system test and acceptance tests. The unit tests and integration tests ensure that the system design is followed in the code. The system and acceptance tests ensure that the system does what the customer wants it to do.

The test levels are planned so that each level tests different aspects of the program and so that the testing levels are independent of each other. Someone is responsible for each level. Each testing level will begin as soon as there is something to test. The traditional V-model states that testing at a higher level is begun only when the previous testlevel is completed. We will not follow this rule, testing in higher levels will begin when the system has been developed to the point where testing at that level can begin. Splitting the testing into V-model test levels that are performed in parallel could mean that the testing process is easily manageable and more flexible than traditional V-model testing. During the project we will try to assess this approach.

The exact features to be tested, the deliverables to be produced, the schedule and the risks to be taken into account will be described in detail in the test plan.

3. Responsibilities

The testing group consists of Markus Jakobsson, Anders Gebala and Björn Forss. The test manager Markus is responsible for testing practises and will be involved in testing at all levels. Testing of unit level code is the responsibility of the programmer himself. Integration testing will be the responsibility of Anders who is also responsible for system integration and regular builds. System testing will be the responsibility of Björn who is also responsible for system architecture. Acceptance testing will be done by the customer. A peer group will also do system testing in the third phase based on the test plan.

Test level	**Responsibilities**
Acceptance tests	Markus Jakobsson, customer
System tests	Markus Jakobsson, Björn Forss, peer group
Integration tests	Markus Jakobsson, Anders Gebala
Unit tests	Programmers

Table 1: Test responsibilities

4. Unit testing

The unit tests are performed on the units and modules while the programmer is coding them. Units and modules committed to the code repository are not allowed to have any known bugs. The programmer of a unit owns this unit and is responsible for testing it and finding any bugs contained in it. The unit tests are written as main methods in the java classes. The method tests the class and its methods. The main method also produces test results as text. This could be seen as a lighter version of unit tests that involve a testing framework.

5. Integration testing

In integration testing the separate modules will be tested together to find weaknesses and bugs in the system. In the ProConf project, integration testing will also include performance testing of the system, including XML-parsing performance, memory consumption and SVG-rendering performance. This way any bottlenecks in the system can be identified and the corresponding module can be redesigned. Anders Gebala will plan and write automatic integration testes, to be used in conjunction with the regular builds of the project. The produced metrics will contain the results of the integration test, for example failed method calls or parsing times. In practice the scripts will run all the unit tests, smoke tests and integration tests that involve many modules. Results of the scripts are written to a logfile.

6. System testing

System testing will compare the system specifications against the actual system. Test cases are derived from the systems use cases, and the functionality of the system is tested based on these tests. System testing will be arranged by Björn Forss, system architect, who also will examine the use of use-cases in the project. System tests will produce metrics that describe the stage of the project. Metrics will be the fail/pass relation for the test cases.

7. Acceptance testing

The goal of acceptance testing is to verify that the user requirements have been achieved. The customer will perform acceptance testing in the evaluation phase of the project, but will also participate in acceptance testing at intermediate stages.

8. Experiences

8.1 Phase I1

During the first implementation phase this testing method was not used very widely, as testing had to wait for test items. Testing was mostly done as unit tests and some integration tests. The method of dividing the testing into V-model levels and having one person responsible for each level did not work as planned. No system tests were done and the integration tests were mainly done by the coders themselves, so Anders Gebala and Björn Forss were not very actively involved in their levels. This will probably correct itself in the next phase when we start using regular builds and start planning system tests. The methods for testing on the unit test level worked very well. Some main methods were isolated as separate classes as they grew rapidly. Overall the testing method worked quite well despite the anticipated low testing activity.

8.2 Phase I2

During this phase testing began to run smoothly. The system had advanced to the level where parts of it fulfilled the user requirements, so testing was simple when matching system functionality directly with user requirements. In this matter the V-model testing method turned out to be quite good. However in this relatively small project, the integration level is easily missed and we did not find it advanteagous to design lots of integration tests when we could test at system level already. But it probably depends on the system one is implementing. When we implement the io-package, integration testing will be important since the io-package will include lots of classes in different packages for loading and saving files. One major change from the plan of this method is that we used all the coders on system test level and skipped a middle hand (Björn Forss). This made the testing more effective and easy to manage when I directly managed the testers. The use cases proved very useful for testing, as

we could use them as test cases directly. However, the use cases must be improved for peer testing phase as the execution details are quite poor, and the peer group has no previous knowledge of the system. To sum up, testing began to function properly and the modified V-model method is working to my satisfaction, although it is rather heavy when managing all three levels (unit to system levels).

8.3 Phase I3

The testing approach described in this document was again altered a little in order to fit smoothly with the coding process. Unit testing was made more automatic, and concentrated on regression testing of gml data structures. No new unit tests where written, so during this phase system testing was the most important level of testing. Time was also running out, so testing was made as efficient as possible though more or less automated regression tests of previously implemented parts, leaving more time for system testing the newly implemented parts. This again proved the full-fledged V-model test method to be heavy and time consuming when the iterations are as small as they have been in this project. Choosing the V-model for its testing levels on the other hand was a good choice, even though we could not use all levels simultaneously in every iteration. The fact that each testing level corresponds to a specification level made planning testing very efficient, as the specifications where already well documented. For example, relying on use cases instead of writing completely new test cases was extremely time saving. The priorities of the user requirements were also used in testing.

The division of responsibilities along the levels proved not to work for efficiency reasons. A single test manager can spread the responsibilities more effectively in a project of this size and get better results. Therefore in this phase all testing was coordinated by the test manager, and Anders Gebalas roll was altered to running the unit regression test suite at the regular builds. The integration level testing has been of little degree in this project, not needing a special responsible manager.

8.4 Phase DE

Testing in the last delivery phase consisted of regression testing on all levels. The use of the altered V-model testing method was used in the same way as before, emphasizing the system level. Some obvious bugs where found during system testing, but the automated unit tests on gml structures also showed some minor bugs that didn’t show up elsewere. This was the first time the regression unit tests was of any real testing value in the project.

During the delivery phase the customer did acceptance testing and discovered alot of bugs that were easy to correct, and verified that the program worked to his satisfaction. During acceptance testing the customer also discovered some features that had not been implemented the way they were supposed to, and this could be fixed. Acceptance testing was quite successful, and the customer used the program to construct gml for part of a power plant.

9 Evaluation of the testing method

During this project the altered V-model testing method described earlier has been used for testing. After a slow start in testing, things started to work in phase I2. The described method had to be further modified in order to fit with this projects budgets and constraints. For example, using a separate test manager for the integration and system test levels proved not to work simply because it would have taken too much time and in reality been much harder to manage. After the modifications the used testing method consisted of running an automated unit test suite, a few ad-hoc integration tests, and system tests on all use cases. Testing was done by everyone except the project manager. Testing was done constantly during the phases, but we also had an intense testing session one or two times per phase to assess the situation of the whole system.
The methods main advantages were its efficiency. Use cases could be used directly as test cases. The method also gave a pretty good picture of how much of the system was implemented, as each passed system tests meant that part of the system was working to customer expectations. A standard V-model would not have worked in this project, as development was iterative, requiring the testing to be iterative to. In each phase we added a bunch of system tests to testing, and in the end all the core features where working and passed testing. The biggest disadvantage of the method used was that testing was done by the coders who implemented the functionality. This was inevitable due to the time budget, because it would have taken too long to show everyone how everything works. Also fixing bugs during testing would not have been possible if someone unfamiliar with the code would have done all the testing. The peer testing was an example of that testing the system without knowledge of its design and implementation was hard. Peer testing results were not of any big value, as the peer tester did not manage to use the system correctly. Due to customer goals, we used testing as a mean to get the system working as it should, and speeding up implementation. Testing discovered bugs in desired functionalities and offered a way to assess the progress of the implementation, but did not concentrate that much on discovering minor bugs that did not affect functionality. Even though fault injection was used it did not play a central role in testing as we were developing a prototype that would validate the gml specification, and not a final end product.
The described testing method in the end became quite tailored for this particular project. As such we cannot recommend it directly for use in another project, but the basic idea of different test levels is certainly applicable in most projects. When properly utilizing the design and specification documents, testing on the different levels can be very effective, and offer a good way of assessing the progress of the implementation.

References

[1] I. Sommerville: Software Engineering, 5th ed, Addison Wesley 1999