



Valuable but vulnerable

Maaret Pyhäjärvi, Senior Testing Consultant at Conformiq
Software in Finland, continues the debate on the V-Model

In the last issue of Professional Tester Geoff Quentin discussed the “Venerable and Vital” V-Model and said that negative opinions on it are “strange”. However, what is actually included in the V-Model seems to be a matter of opinion, although the simple form in which the model is presented may suggest it is generally accepted knowledge. In addition, it provides little support for the actual integration of development and testing in an effective and efficient manner.

This article is based on the experiences of some thirty Finnish software companies.

What does the V-Model actually mean?

Both praise and criticism of the V-Model stem from the fact that one can interpret it in many ways. We find some of the intuitive interpretations misleading; our experience has been that we’ve been able to best avoid the negative interpretations by combining the left-hand side onto one box and the right-hand side onto another and treating development and testing as equals in that sense. In our experience many of the problems the model has caused come from the approach of having, typically, three to five set levels that may be interpreted and implemented in various ways. The set levels provide a structure of looking at different things on different levels, but the agreement on what each level would actually contain in practice is not clear.

In a discussion on the context-driven testing list (<http://groups.yahoo.com/group/software-testing>), James Bach summarized the heuristics in the V-Model:

- if you want to get something you want, it may help to know what you want, first
- if you want to build something, especially something that is complex, it may help to have a physical blueprint laid out in advance
- it may be better to finish a given task before starting work on another task that depends on the output of the first task

- it may help to produce blueprints for each aspect of the product before building the product itself
- it may help to think about a product in terms of a hierarchy of blueprints as follows: requirements, external design, internal design, modules
- it may help to test each blueprint, as well as to use each blueprint as a basis for a test process, as the end product is being built.

Only the last one of these heuristics is actually related to testing. They nicely summarize what many people actually see in the V-Model: the waterfall model on the left-hand side and thus not an effective way to develop software in most contexts. And that the simplest interpretation of the model is to see it as communicating testing against each document in the documentation hierarchy.

We believe the V-Model and its variations essentially bring forth three important points:

- 1 testing a smaller part before putting it into a larger system is a good approach
- 2 there are several points of view to test for
- 3 testing efforts can start with planning as soon as the higher level requirements have been identified.

This does not yet provide much structure for testing, but reminds us of some of the essential concepts. We view the V-Model as a mechanism for communication in negotiating who would do what in testing, not a model of how it must be done. However, we’ve found it common to be used as a rationale of what documentation there must be for testing as well as what others should be doing in their testing phases, even though this interpretation is more often a negative one.

The scheduling effect

Development models consider testing as an afterthought [1,2] and the V-Model as well as its variations seem still just extensions of the waterfall model. Even in organizations that do not follow the waterfall model, it has been common to view the project through the

V-Model and end up with what we call “the scheduling effect” of the V-Model that essentially stems from the waterfall assumption. Even if the implementation is structured differently, testing may still be structured with the right-hand side of the waterfall. It has been claimed that the negative sides of waterfall assumption embedded in the V-Model has been avoided in Europe, or at least in the UK [2]. However, in our experience, this is the pitfall students on our courses face and it is extremely common in companies as well, as the model is used to structure separate group’s testing, not testing in the development lifecycle.

The V-Model used as a basis for creating schedules results in defining a testing phase – a reserved time for the activity – for each of the levels. This is also implied in the choice of words in the ISEB Foundation Level Certificate in Software Testing syllabus, where the levels are referred to as “stages” – implying both serialization and different focus. The levels should, however, be interpreted as continuous activity with a perspective, as seemed to be the case with the earliest reference we’ve seen to the V-Model in [3]. This continuity – testing as a service – is especially true if any of the levels is a “separate testing group’s responsibility”. The form in which the V-Model is presented communicates the idea of continuity poorly. These phased structures based on the form of the model are, in our experience, common and inefficient, as well as difficult to change.

To make the scheduling effect more tangible, we have opened the V-Model to two concurrent streams of activities (see figure 1). The Implementation stream flows in waterfall fashion from Requirements to Coding and during test execution phases the debugging and fixing of defects found takes place. The testing stream flows from planning each test level to executing each level of tests. Consider a defect introduced in the Requirements phase. The tests to catch this defect would be scheduled at the end of the project with potentially large amount of avoidable rework. Finding and fixing a software problem after delivery is often 100 times more expensive than finding and fixing it

If you ignore these opportunities, it's not just software you should be testing!

£ATTRACTIVE + EXCELLENT BENEFITS + RAPID CAREER PROGRESSION

Tescom is a leading international Information Technology (IT) Quality Assurance (QA) service-provider with more than 600 professionals around the globe - including 100 in the UK - dedicated to the value enhancement of IT. Since 1990, Tescom's business is mitigating the inherent technological risks of software and IT applications development, integration and implementation.

Among Tescom's solution offering are Project Evaluations and Audits, from which can be derived specific IT control processes, including classic Testing services. Tescom also provides Configuration Management services. Tescom has very ambitious plans for expansion in the UK and is seeking equally ambitious people who thrive in a fast-paced, dynamic work environment, with the opportunity to garner international experience.

CALLING ALL TESTING PROFESSIONALS! TEST ANALYSTS, SENIOR TESTERS, TEAM LEADERS & TEST MANAGERS

Requirements include a minimum of two years relevant experience, good client and consultative skills, the personality and flexibility to thrive as part of a team in a global organization as well as the ability to learn quickly and the drive to get things done. Tescom is particularly interested in people whose skill set includes any of SMS, MMS WAP, Mercury and Rational Suite, Biztalk, iDTV, Oracle Applications, Citrix, Radia and Peoplesoft.

Tescom also has opportunities in Denmark for people with knowledge of Unix, SQL and Test Director.

Tescom's exceptional client base, variety of new projects and range of sectors mean the opportunities for personal and professional growth with Tescom are immense - and we are dedicated to rapid promotion from within.

So why not join us? And test yourself to the full.

Please send your CV to jobsuk@testcom-intl.com or post it to
Tescom, Attention: HR Department, 21-22 Great Sutton Street, London EC1V 0DN.

during requirements and design phase [4] and acceptance testing is the final check prior to production. Reluctance to fix eg requirement bugs found in acceptance testing or specification bugs found in system testing, as these are seen as the V-Model waterfall's last phases, places testers as change recommenders late in the project, when every change they recommend will carry a significant price [5].

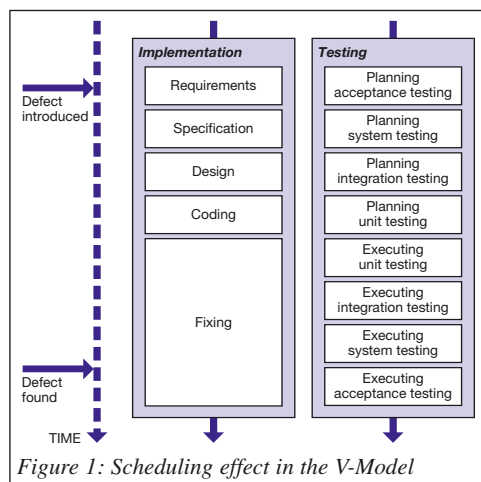


Figure 1: Scheduling effect in the V-Model

The appropriate level of planning for testing should include planning the test cases, and this is said to be an effective way of finding defects [2]. However, writing them down before being sure the requirements will actually be implemented results in wasted time. The approach in planning should rather include reviews as the more cost-effective alternative to writing test cases that will never be executed [5]. One could argue that these reviews are an essential part of test planning and thus mitigate the negative effect on the cost to correct a defect in this interpretation of the V-Model. However, while peer reviews catch 60% of defects [4] and improving reviews to be perspective based catch 35% more defects [4], the resulting percentage is still only 80%. And since defects are not of the same value, but 80% of avoidable rework comes from 20% of defects [4], what guarantees do we have that reviews will catch at least some of the expensive 20%? In addition to this, the waterfall assumption of no changes is likely to be incorrect, and there is a constant stream of requirements to review.

An approach we have seen testers suggest to correct the potential cost due to distance between defect introduction and defect finding is to shorten the length of the iteration. However, repeating all the test levels in the same fashion in scheduling adds to the overhead and brings forth new challenges in regression testing. Also, it is difficult to develop meaningful content for an upcoming release in short increments.

Another scheduling implication from the V-Model interpreted as above is that finding bugs and re-testing and regression testing is typically not planned for. For each of the phases, there should be at least one round of

re-testing and regression testing and typically this one round is not enough.

One important context in which the V-Model is considered to be appropriate is contracted software. If a customer is likely to change his mind and unlikely to willingly take responsibility of the costs associated with the changes, the V-Model's built-in waterfall creates a clear contract with clear rules for allocation of risk [5]. After signing off requirements, every change request is a scope change associated with a cost to the customer. This may not serve the customer best and for in-house development, the cost structure is not favourable.

In scheduling, the approach we have been hoping for is to enable planning of tests based on all levels of requirements and planning for executing these tests in the order that makes most sense, instead of the order implied in the V-Model. As Parnas notes [6], in using a model such as the waterfall - or the V-Model - it is essential to understand in the first place that you are actually faking the serialization only to clarify the different roles with the help of the different phases. We've found it helpful to think of two types of testing: in-synch testing comprises the testing tasks that can be done at the same pace as the development proceeds, in continuous fashion, and off-synch testing refers to the typical view of planned and prepared testing that takes too much time to be integrated into the pace of the development. These concepts have been helpful in planning for both continuous service of the separate testing group as well as communicating the end-of-project's testing efforts that are on the critical path to release.

Structuring technical and end-user testing

We've found scaling up from a small number of test levels easier than scaling down from the most common four levels. It has been our experience that we need at least two levels of testing: the technical perspective (typically developer testing) and the end-user perspective (typically system/acceptance testing). If we have a system of systems, involving many groups, we typically would add more levels to use. We've found it difficult to separate unit and integration testing for the developer as it all just relates to an implementation task at hand. It has also been difficult to separate continuous, incremental system testing from integration testing in practice.

In our efforts to make the levels of the V-Model more manageable, we typically re-define the levels with respect to several dimensions we feel that need to be agreed on, even if some people may interpret them already included in the model. The V-Model describes test levels, which refer to different levels of abstraction, and the size of the target in testing. To add to this, we need the concept of test types, and test complexity levels. It is important to understand that these concepts refer to different dimen-

sions of testing. Test complexity level is used to describe the complexity and depth of the designed and executed test cases. The tests need to grow more complex as the product matures. Test types remind us of the different aspects of the system we need to test for.

For each test level, we typically discuss and define:

- who is responsible for the level of testing
- who plans the level of testing
- who executes the level of testing
- what is expected of regression testing on the level
- test execution timeframe (in-synch/off-synch) and its relative management mechanism (agile/plan-driven)
- test environments used
- perspective (technical vs end-user)
- test complexity levels expected (typically smoke/function/capability/complex)
- test types
- test techniques and targeted defects
- documents to base testing on.

Closing thoughts

The V-Model does not yet provide much structure for testing, but reminds us of some of the essential concepts which have a significant impact on the success and maturity of testing. It provides a mechanism for communication in negotiating who would do what in testing, not a model of how it must be done. Understanding the way each of us uses the model and what assumptions we have built helps in mitigating the inherent risks of people assuming things we would not want from the V-Model's form.

References

- [1] Marick, B. New Models for Test Development. 1999. Proceedings of Quality Week 1999.
- [2] Goldsmith, R. F. and Graham, D., "The Forgotten Phase," Software Development, 2002.
- [3] Boehm, B. Guidelines for Verifying and Validating Software Requirements and Design Specifications. Samet, P. A. EURO IFIP 79, 711-719. 1979. North-Holland Publishing Company.
- [4] Boehm, B. and Basili, V., "Software Reduction Top 10 List," Computer, vol. 34, no. 1, pp. 135-137, 2001.
- [5] Kaner, C., Bach, J., and Pettichord, B., Lessons Learned in Software Testing - A Context-driven Approach Wiley Computer Publishing, 2002.
- [6] Parnas, D. L. and Clements, P. C., "A Rational Design Process: How and Why to Fake It?," IEEE Transactions on Software Engineering, vol. 12, no. 2, pp. 251-257, 1986.