Multiple V-model in relation to testing

Sogeti Nederland B.V.
Edwin Notenboom
edwin.notenboom@sogeti.nl

In embedded systems, the test object is not just executable code. The system is usually developed in a sequence of product-appearances that become more real. First a *model* of the system is built on a PC, which simulates the required system behavior. When the model is found to be correct, code is generated from the model and embedded in a *prototype*. The experimental hardware of the prototypes is gradually replaced by the real hardware, until the system is built in its final form as it will be used and mass produced. The reason for building those intermediate product-appearances is, of course, that it is cheaper and quicker to change a prototype than to change the final product, and even cheaper and quicker to change the model.

The multiple V-model, based on the well-known V-model is a development model that takes this phenomenon into account. In principle each of the product appearances (model, prototypes, final product) follows a complete V-development cycle, including design , build and test activities. Hence the term 'multiple V-model' (see Figure 1). The essence of the multiple V-model is that different physical versions of the same system are developed, each possessing the same required functionality in principle. This means, for instance, that the complete functionality can be tested on the final product. On the other hand, certain detailed technical properties cannot be tested very well on the model and must be tested on the prototype - for instance, the impact of environmental conditions can best be tested on the final product. Testing the different physical versions often requires specific techniques and a specific test environment. Therefore a clear relation exists between the multiple V-model and the various test environments.

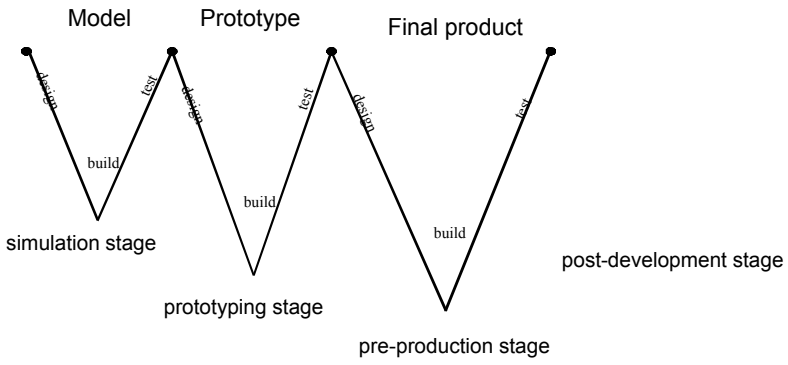

Figure 1: Multiple V-development lifecycle

The test process involves a huge number of test activities. There are many test design techniques that can and will be applied, test levels and test types that must be executed, and test related issues that require attention. The multiple V-model assists in structuring these activities and issues. By mapping them onto the multiple Vs, it provides insight to the questions: "When can activities best be executed?" "Which test issues are most relevant at which stage in the development process?"

Organising the testing in the complex situation of multiple V-model development is itself a complex task. In order to plan and manage this well, the test manager needs an overall picture of all relevant test activities and issues onto the multiple Vs. It takes a broad range of activities and issues that are to be considered. They are in alphabetical order and divided into three categories: test techniques; test levels and types; and other issues. Each is then put on one or more Vs at the relevant stage of that V. It is possible for certain issues, such as low-level requirements, to appear in the V for the model as well as in the V for the prototype.

The multiple V-model with the three sequential V-cycles does not take into account the practice of (functional) decomposition of a complex system. The development of such a system starts with the specification of the requirements at a high-level, followed by an architectural design phase where it is determined which components (hardware and software) are required to realise this. Those components are then developed separately and finally integrated into a full system. In fact the simple V-model can be applied to this development process at a high-level. The left side of the V-model handles the decomposition of the system into its components. The middle part of the V-model handles the decomposition of the system into its components. The right side of the V-model handles the integration of the components.

This principle can be repeated for components that are too big or too complex to develop as one entity. For such a component, an architectural design activity is carried out to determine which subcomponents are required. Because this may result in a V-model within a V-model the development lifecycle model is said to be nested.

In fact the purely sequential multiple V-model is mainly applicable to the component level. Usually it

is not the complete system which is fully modeled first, then completely prototyped, and so on. It is the components that are built in this stepwise way. This explains why some development activities and issues cannot be mapped very well on the 3 Vs in the multiple V-model - for instance high-level and low-level requirements, safety plan, design and build specific tools. That is because they belong to the overall development process.

When the V-model at the system level is combined with the multiple V-model at the component level, it results in the so-called 'nested multiple V-model' (see Figure 2).

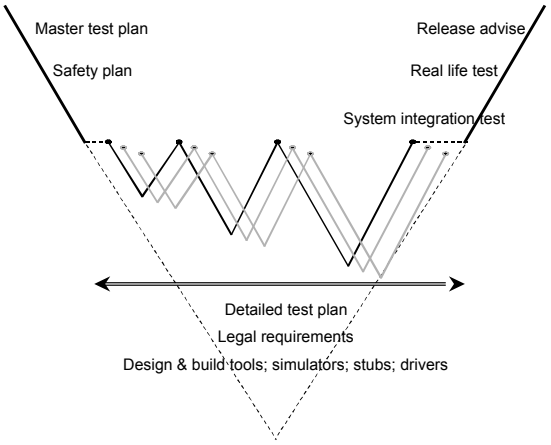

Figure 2: Higher level test issues in the nested multiple-V model

If every test level would define for itself the best things to do, there would be a significant risk that some tests would be performed twice and others would be omitted. In practice, it is not unusual for the system test and the acceptance test to have the same test design techniques applied to the same system specification - this is a waste of valuable resources. Another risk is that the total test process, spends longer on the critical path than is necessary because the planning of the different test levels has not been coordinated. Of course it is better to coordinate the various levels: it prevents redundancies and omissions; it ensures that a coherent overall test strategy will be implemented. Decisions must be made as to which test level is most suited to testing which system requirements and quality attributes.

A master test plan can be viewed as the combination of what has to be tested and who is going to perform the test activities. After completing the master test plan, for each test level the staff involved will know exactly what is expected of them and what level of support and resources they can expect. The master test plan serves as the basis for the detailed test plan of each test level.

The development of a system containing embedded software involves many test activities at different stages in the development process, each requiring specific test facilities. Some organisations have a testing process divided into test as shown in figure 3. This can be mapped onto the development stages (see Figure 1). The early models developed to simulate the system are tested in 'model tests' (MT) and 'model-in-the-loop' (MiL) - this corresponds to the simulation stage. In 'rapid prototyping' (RP) an experimental model with high performance capacity and plenty of resources is tried out in a real-life environment to check if the system can fulfill its purpose - this is also part of the simulation stage. In 'software-in-the-loop' testing (SiL) the real software including all resource restrictions is tested in a simulated environment or with experimental hardware. In 'hardware-in-the-loop' testing (HiL) the real hardware is used and tested in a simulated environment - both 'SiL' and 'HiL' are part of the prototyping stage. In 'system test (ST) the real system is tested in its real environment - this corresponds to the pre-production stage.

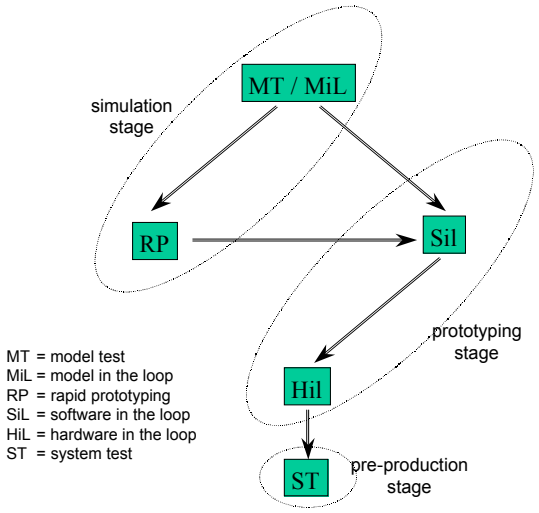

Figure 3: A test process showing the gradual transitions from “simulated” to “real” system