ASSESSING AND IMPROVING SOFTWARE QUALITY IN SAFETY CRITICAL SYSTEMS BY THE APPLICATION OF A SOFTWARE TEST MATURITY MODEL

F.I. Duncan*, A.G. Smeaton [†]

**Director BitWise Ltd, UK papers@bitwisegroup.coms, [†]CTO BitWise Ltd, UK papers@bitwisegroup.com*

Keywords: Software, quality, test, maturity models, safety.

Abstract

Drawing on wide experience over many years, BitWise has evolved a SOFTWARE TEST MATURITY MODEL. This is of particular value in the testing of safety critical software. This brings significant benefits in terms of cost and effective quality. This paper explains the Model and enables development groups to assess their current capabilities and plan any required improvements.

1 Introduction

Businesses developing safety critical software will reasonably be expected to have a set of well defined processes. Some processes will be supported by tools (commercial, open source or bespoke) to improve efficiency and reduce risk of human error. Continuous improvement is generally driven by either actual failures in the processes (leading to piecemeal additions to plug the gaps) or regulatory changes. Whatever the drivers, tools and processes are often introduced without regard for overall effectiveness.

Periodically, BitWise is asked by clients to assess their current processes and tools and to recommend the best way to achieve improvements. It is quite common to find a complex mixture of processes, practices and tools but little or no understanding of their overall effectiveness. It can also be challenging to find a simple and consistent basis for assessing current systems and formulating an improvement plan that can be delivered in a phased manner.

BitWise has evolved a maturity model approach which has proven to be very effective in addressing this challenge.

The novelty in this paper lies in the specific methods of application and their relationship with wide real experience. BitWise makes no claims of originality with the general maturity model concept [1] or its application to software testing [2].

2 Scope of model

The safety lifecycle from IEC61508 [3] is shown in Figure 1. It shows the overall scope of the safety lifecycle. Phase 10 is where the software development and test takes place so is of particular relevance to this paper.

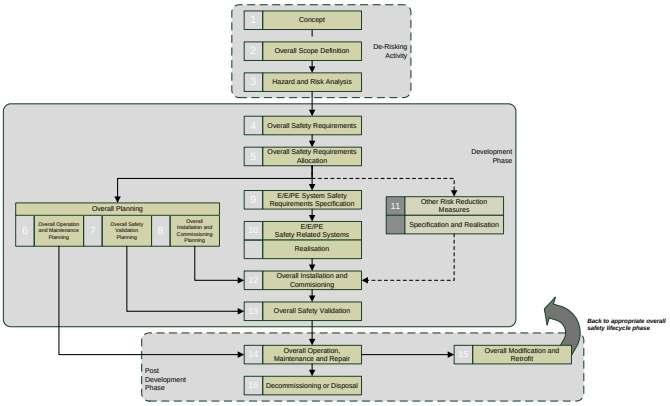

Figure 1 Safety lifecycle from IEC61508

Phase 10 of the IEC61508 safety lifecycle breaks out into a more detailed software lifecycle. This is summarised in Figure 2.

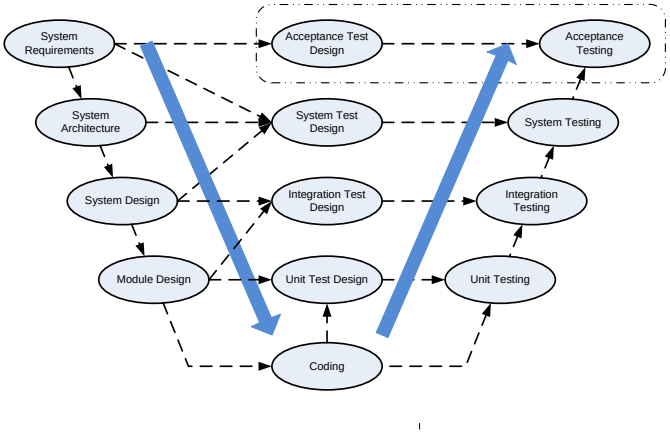

Figure 2 Software development lifecycle

The focus of this paper is on software testing. So the scope of the BitWise model is focused on phase 10 of the safety lifecycle i.e. the various phases of the V model shown in Figure 2. Critically, although testing appears explicitly on the right hand leg of the V model, testing is impacted by all

phases of the lifecycle. The Model ensures that each phase is consistent with the level of testing required.

The Model could potentially be extended to other phases of the safety lifecycle. However, this paper focuses on the software development lifecycle.

2.1 Quality gates

Each phase in the software development lifecycle can and should have a set of quality gates at its end. Additional quality measures also need to be applied during the phase. There are a number of mantras that are often applied in selecting quality measures for each gate:

1. Issues should be caught at the earliest point. The cost of fixing an issue grows exponentially in later phases.
2. Phase escape will happen so later phases need to be robust.
3. System testing is often the last line of defence.

The Model adds an additional key point.

4. There needs to be a balance of measures across the various phases. An imbalance can easily lead to major overspend without significant benefit.
5. Testing needs to be considered at all phases of the lifecycle.

Quality measures vary from phase to phase. There are many techniques and tools that can be applied.

3 The Model

The Model is based on five levels of increasing test maturity. These are introduced first before looking at what techniques and tools apply to each level. The first two levels are very basic and it is reasonable to expect any business developing safety critical software to be at least at level three.

3.1 Maturity levels

There are five maturity levels in this model. These are characterised as follows:

Level 1 - Initial

- A chaotic process. There are little or no written processes
- Not distinguished from debugging and ill defined
- The tests are developed ad hoc after coding is complete
- Usually lack a trained professional testing staff and absence of testing tools
- The objective of testing is to show that the system and the software work

Level 2 – Phase Definition

- Identify testing as a separate function from debugging
- Testing becomes a defined phase following coding
- Processes documented/standardised to the point where basic testing techniques and methods are in place
- The objective of testing is to show that the system and software meets specifications

Level 3 – Integration

- Integrate testing into the entire life cycle
- Establish a formal testing organisation
- Establishes formal testing technical trainings
- Controls and monitors the testing process
- Begins to consider using automated test tools
- The objective of testing is based on system requirements
- Major milestone reached at this level: management recognises testing as a professional activity

Level 4 – Management and Measurement

- Testing is a measured and quantified process
- Development products are now tested for quality attributes such as Reliability, Usability, and Maintainability
- Test cases are collected and recorded in a test database for reuse and regression testing
- Defects found during testing are now logged, given a severity level, and assigned a priority for correction

Level 5 – Optimisation/Defect Prevention and Quality Control

- Testing is institutionalised within the organisation
- Testing process is well defined and managed
- Testing costs and effectiveness are monitored
- Automated tools are a primary part of the testing process
- There is an established procedure for selecting and evaluating testing tools

3.2 Test maturity model effects

As successively higher levels of the Software Test Maturity Model are attained, the effectiveness and efficiency of the quality measures will improve. The following effects are typical of the improvements noted when adopting this Model.

Effects of operating at Level 3

- Fewer defects are found during testing activities.
    - Institutionalised peer reviews and inspections find defects early.
    - More formalized engineering process focus on quality requirements and design documents.
- There is a significant reduction in the time spent conducting system and user acceptance testing.
- Testing efficiency and effectiveness are known (measured and analyzed), and used for future project planning.
- Test results are used to predict project milestone completions.

- Automated test systems are evaluated and considered for implementation.

Effects of operating at level 4

- Fewer defects are found during later testing activities.
- Most defects are found during peer reviews and early testing activities.
- Test completion criteria can be based on quantitative data from tests conducted.
- Automated testing support is built into product.

Effects of operating at level 5

- Fewer defects are found during testing activities. Improved engineering activities prevent defects from being built into the work products.
- Prototyping activities help eliminate potential requirements and design flaws.
- Entire testing levels may be dropped, or combined with others.
- Automated testing tools replace previously manual processes.
- Testing efficiency and effectiveness are improved.

4 Model details

The Model is represented as a two dimensional matrix. One dimension is the maturity level and the other is a list of quality measures. The latter are organised in lifecycle phase order.

Each quality measure can take on one of two forms. The first form is where there are a number of different levels of application each of which is assigned to a different maturity level. The second is a simplified form where there is only one level of application i.e. it is either applied or it is not. So each cell in the matrix is either a simple yes/no or a more detailed definition of the particular level of application.

There are many dependencies between different quality measures. A large part of the Model development effort has been in identifying these dependences and ensuring that the level definitions do not have any dependency on something at a higher level.

This paper does not provide the full detail of the matrix. The matrix is augmented as new tools and techniques become available. Indeed, the matrix will continue to evolve as BitWise applies it in new situations.

A good example is effective Unit testing. These rely on detailed interface specifications coming from the module design phase of the lifecycle. So unless the design work is mature enough, the project has to rely on integration testing and largely bypass unit testing. Going a stage further, the definition of components needs to be clear. So unless there is a well define modular architecture with clear identification of components, it is very difficult to identify component interfaces.

5 Using the Model

The Model can be applied at two levels. Firstly it can be applied to a development organisation to assess its overall test maturity level. However, individual projects typically adopt different quality plans. So this Model is even more useful to assess and find improvements to the test strategy for an individual project.

Using the Model is relatively straightforward. Some independent assistance is generally required in assessing the actual level of some processes, techniques and tools, particularly where they are bespoke.

A copy of the matrix is marked up to show which level each measure is currently at. Once this is complete, it is relatively simple to form a view on the current maturity level.

In an ideal world, there should be a high level of consistency i.e. every quality measure is at the same level. In practice, there is normally a spread. Typically, there may be a cluster around level 3 or sometimes level 4. However, there will be a few below level 3 and these are clearly the top priority action points. There are also some quality measures up at level 5, often carrying considerable cost and adding little benefit.

The action plan is relatively easy to prioritise by focusing on each level of maturity in turn. This can become more complex when there is a mix of say level 3 and level 4 capability. There could be a substantial number of issues to address to bring the business fully up to the higher level. A further level of prioritisation still needs to be applied but that has to be done on a case by case basis.

6 Conclusion

BitWise has applied this Model to its own processes. It has used this to consolidate at level 4 and establish a plan to progress to level 5. BitWise also applies the Model to each project to ensure consistency of the quality plan.

The Model has also been applied to a number of client processes and their key projects. It has proved very useful in developing improvement plans and setting priorities.

The Model continues to evolve but has already proven to be a very useful technique.

References

[1] Carnegie Mellon University, Capability Maturity Model for Software, version 1,1, CMU/SEI-93-TR-024-ESC-TR-93-177 (1993)

[2] J D Hart, Software testing and the Capability Maturity Model, Innovation Dynamics Consulting, (2000)

[3] International Electrotechnical Commission, Functional Safety of electrical/electronic/programmable electronic safety related systems, Parts 1 to 7.IEC ISO/IEC 61508 (2010)