Agile Development of Safety-Critical Software for Machinery:

A View on the Change Management in IEC-61508-3 ed2.0

Jani Paalijärvi (currently with Nomovok)

Tampere University of Technology

Mika Katara

Tampere University of Technology

With help from:

Mika Karaila

Metso Automation Inc

Teemu Parkkinen

Sandvik Mining and Construction Oy

This presentation, based on the first author's MSc Thesis, reports results of the OHJELMATURVA project. Partial funding from Tekes and the companies participating in the project is gratefully acknowledged.

TAMPERE UNIVERSITY OF TECHNOLOGY

Safety Critical Software in Machinery

- There is a trend towards using software to implement features that have been traditionally implemented in hardware
- New complex features are implemented using software
- Benefits of software compared to hardware include:
  - More advanced functionality without a need to increase the size and the capacity of the hardware
  - Especially in large product quantities software can provide cost savings
  - Flexible changes are supported without having to change physical parts
  - Large amount of information about the system and its performance can be gathered for monitoring etc. purposes

- Machinery is often safety critical
- Software safety is part of the overall safety of the machine
- Software can increase system safety, but can also cause accidents in the case of errors
- The role and interactions of software should be understood when applied in safety critical systems
- In contrast to hardware and mechanics, there are no tolerances or safety margins
- The concept of reliability is fundamentally different for software and hardware: since the former does not wear, our confidence towards a software component increases while in use for a long time without failures (“proven in use”)
- Software safety is assured by examining the development process in additional to verification & validation

IEC 61508-3 ed2.0

While the lifecycle model of IEC 61508-3 ed2.0 is based on the traditional V-model type of development, organizations developing software for machinery are moving towards more agile and lean software development processes and practices

In practice, there is a need to tailor the development process to be more flexibility, but still satisfy the requirements of the standards

Obviously, such tailoring must be justified on the basis of functional safety

This is challenging because there are obvious ideological differences between the standard and agile and lean practices

For instance, the former requires heavy documentation, while the agile principles promote undocumented face-to-face communication

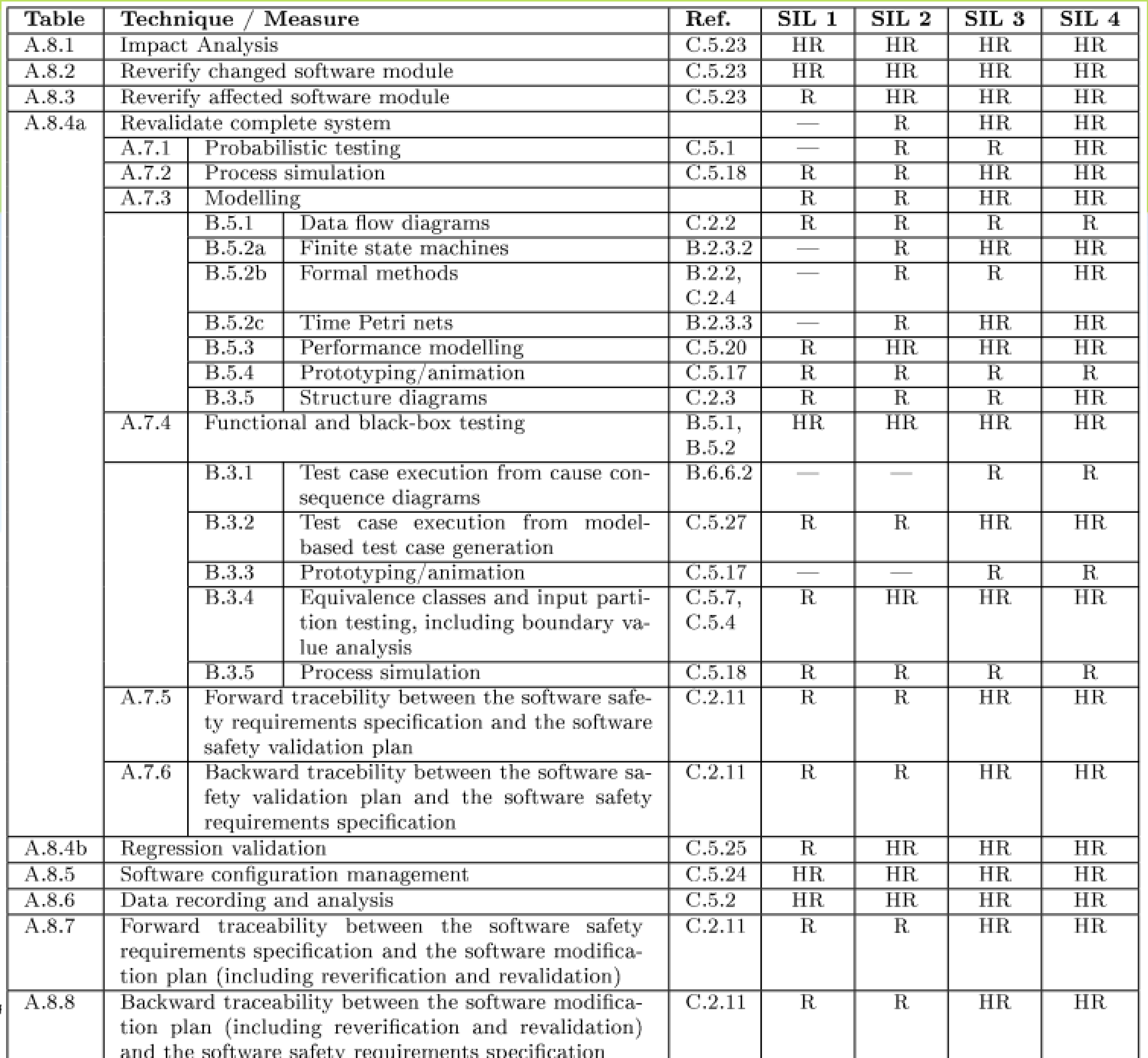
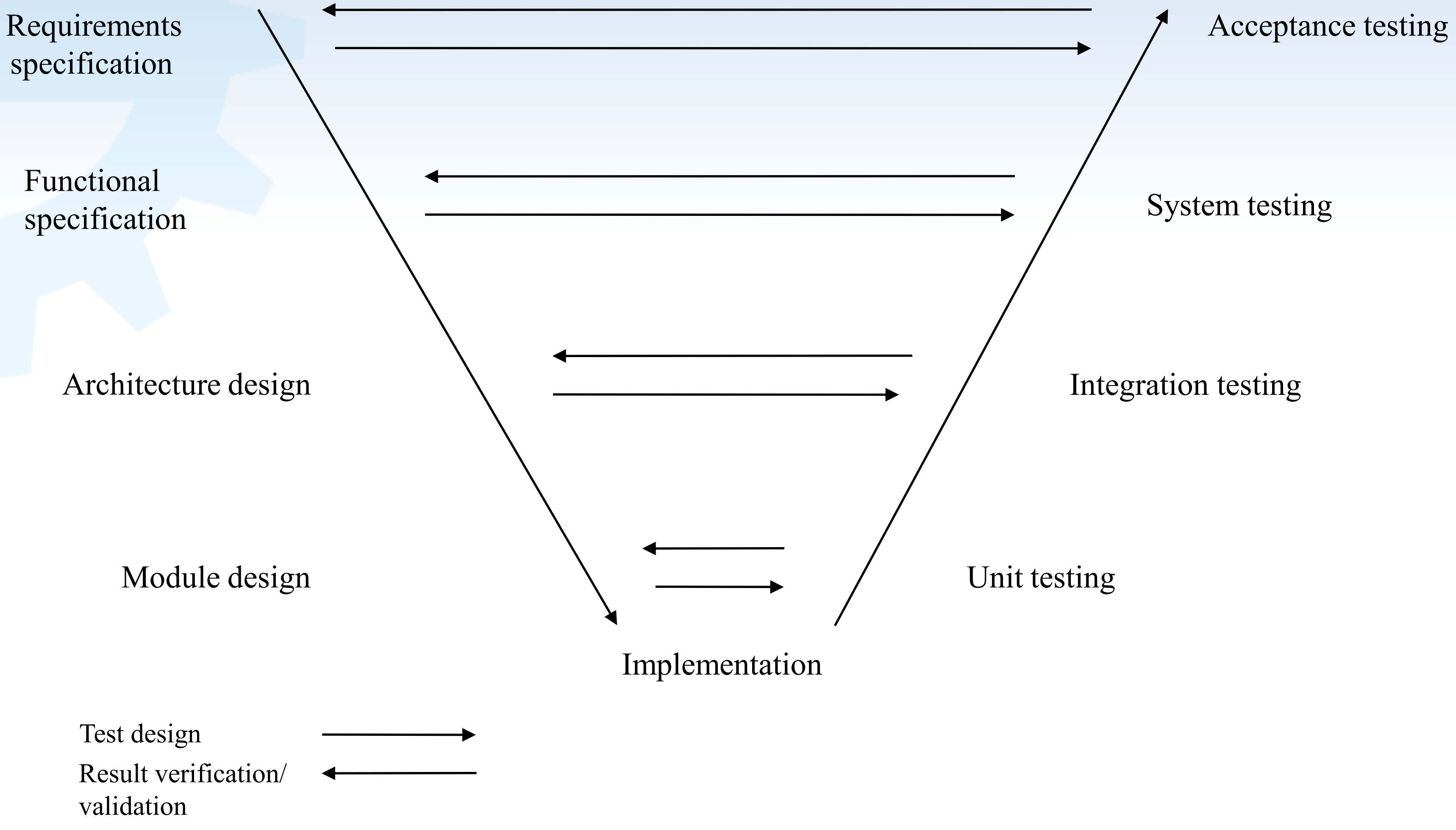


Table	Technique / Measure			Ref.	SIL 1	SIL 2	SIL 3	SIL 4
A.8.1	Impact Analysis			C.5.23	HR	HR	HR	HR
A.8.2	Reverify changed software module			C.5.23	HR	HR	HR	HR
A.8.3	Reverify affected software module			C.5.23	R	HR	HR	HR
A.8.4a	Revalidate complete system				—	R	HR	HR
	A.7.1	Probabilistic testing		C.5.1	—	R	R	HR
	A.7.2	Process simulation		C.5.18	R	R	HR	HR
	A.7.3	Modelling			R	R	HR	HR
		B.5.1	Data flow diagrams	C.2.2	R	R	R	R
		B.5.2a	Finite state machines	B.2.3.2	—	R	HR	HR
		B.5.2b	Formal methods	B.2.2, C.2.4	—	R	R	HR
		B.5.2c	Time Petri nets	B.2.3.3	—	R	HR	HR
		B.5.3	Performance modelling	C.5.20	R	HR	HR	HR
		B.5.4	Prototyping/animation	C.5.17	R	R	R	R
		B.3.5	Structure diagrams	C.2.3	R	R	R	HR
	A.7.4	Functional and black-box testing		B.5.1, B.5.2	HR	HR	HR	HR
		B.3.1	Test case execution from cause consequence diagrams	B.6.6.2	—	—	R	R
		B.3.2	Test case execution from model-based test case generation	C.5.27	R	R	HR	HR
		B.3.3	Prototyping/animation	C.5.17	—	—	R	R
		B.3.4	Equivalence classes and input partition testing, including boundary value analysis	C.5.7, C.5.4	R	HR	HR	HR
		B.3.5	Process simulation	C.5.18	R	R	R	R
	A.7.5	Forward tracebility between the software safety requirements specification and the software safety validation plan		C.2.11	R	R	HR	HR
	A.7.6	Backward tracebility between the software safety validation plan and the software safety requirements specification		C.2.11	R	R	HR	HR
A.8.4b	Regression validation			C.5.25	R	HR	HR	HR
A.8.5	Software configuration management			C.5.24	HR	HR	HR	HR
A.8.6	Data recording and analysis			C.5.2	HR	HR	HR	HR
A.8.7	Forward traceability between the software safety requirements specification and the software modification plan (including reverification and revalidation)			C.2.11	R	R	HR	HR
A.8.8	Backward traceability between the software modification plan (including reverification and revalidation) and the software safety requirements specification			C.2.11	R	R	HR	HR

Software Development Methods

- Plan driven process models are usually based on waterfall and V-model type of processes
- While using the V-model can provide effective means for validation & verification, it has some well known drawbacks:
  - If software is integrated and integration tested late in the development cycle, problems will appear and they will cause delays in delivery schedules
  - If the customer sees the software in action only in acceptance testing phase, changes to meet the actual needs are no longer possible
  - Often in practice, the requirements for the software are frozen during the project, not in its beginning


Requirements specification
Acceptance testing
Functional specification
System testing
Architecture design
Integration testing
Module design
Unit testing
Implementation
Test design
Result verification/ validation

When Winston W. Royce introduced the waterfall model, he already suggested using iterative and incremental features

Later, there have been many variations of iterative and incremental process models

Agile methods are those which more or less conform to the ideas introduced in the agile manifesto:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

Pros and Cons of Plan-Driven Methods

- Pros: easy to grasp, clarity, lots of experiences and tool support
  - Pre-study phase enables giving price and schedule estimates
  - V-model requires well planned and continuous testing
- Cons: does not support heavy changes, assumes “perfect” requirements
  - Estimates on price and schedule often prove to be wrong
  - Tries to solve the problems in too big chunks
  - Bears big risk on producing something that does not correspond to the needs of the customer or end-user

Pros and Cons of Agile Methods

Pros: flexibility, “embracing changes”, do not rely on “perfect” requirements, close collaboration with the customer

Cons: “Working software over comprehensive documentation” is usually interpreted as “working software is enough, no need to document anything” since developers don’t like to write documentation and documents are burdensome to keep up-to-date

The role of software architecture maybe easily overlooked, unless the method used places special emphasis on that

Too much hype, too little experiences on what works and in which contexts

Requires skillful staff and a customer who is available on a daily basis

Embracing changes may lead to short-sighted solutions

Development of Safety Critical Software

- Many of the errors in safety-critical systems can be traced back to requirements
  - Understanding the interplay between the requirements and safety requires good understanding of the application domain
- Safety-critical systems are often embedded systems
  - If the hardware is developed concurrently with the software, it can cause changes to the software requirements
- Another typical type of problem relates to interfaces, for instance between the hardware and software, which can be caused by problems in communication and misunderstandings
  - Communication between the teams is important so that the information about the changes is propagated to everyone

- In the literature, there are many different views on how well agile and lean methods suit for creating safety-critical products
- Reported experiences in applying agile methods at least in the context of DO-178B standard for airborne software
- Problems related to light design and documentation, refactoring, and the lack of systematic working methods
- Also the interplay between software architecture and design is a problem in the case of frequent changes
- Refactoring the architecture of large and complex systems can be hard and it requires above average skills from the developers

- Refactoring the architecture and code can make the system clearer and more understandable
- Extensive regression testing and continuous integration can mitigate the risks involved in refactoring
- In safety-critical systems it might be necessary to prohibit the refactoring of large chunks at a time
  - “License to refactor” can also be limited to certain developers only

- Collaboration, incremental and iterative development are pros
  - It is important to share understanding on the requirements
  - Discussing a requirement within the team can be very productive
- In practice, some hazards can be overlooked in the beginning of the project, thus, they should be reviewed in iterations in the light of new knowledge gained in the project
- Agile teams are in charge of the development, it is thus vital that the team understands its responsibility, and is committed to following the rules instead of the “the way of hacking “
  - In some cases, the self organizing nature of the teams may have to be limited?

- Special emphasis is needed to make sure that the work is done is a systematic and accurate manner, and that documentation and architecture are emphasized as well as the clarity and traceability of the implementation
- However, some of the agility may be lost
- This can be alleviated by:
  - Test automation
  - Automatic document creation
  - Requirements management
  - Using tools to trace requirements
  - Making reviews more effective
- Due to the burden or re-verification and re-validation, it is vital to limit the impacts of changes during the whole project

Conclusions

- Current agile software developments methods are not applicable to the development of safety-critical software as such
- However, mixing some of the principles of these methods with some formal techniques may provide a satisfactory solution
- Balancing between these two worlds can be difficult
- If the organization is already using V-model type of development, agility can be increased with the following, for instance:
  - Iterative and incremental process
  - Close collaboration and communication with the customer/end-user and between the teams
  - Tools for life-cycle management of requirements and traceability
  - Test automation
  - Continuous integration
  - Implementation and architecture that are clear and modifiable

- Documentation can be partly automated using tools, but there is no way to escape the burden completely
  - Maintenance and reviews of documents need special attention
  - Documentation should probably be limited to just meet the requirements of the standards
- Agile methods need new skills and ability to work in small teams
- It is probably wise to start with the V-model type of process, trying to include some agile and lean principles one by one, rather than the other way around
- While the standard requires some ordering on tasks based on the V-model, the tasks should be assigned to iterations
  - In practice, not all the required tasks need to be done in all iterations
- This emphasizes the role of planning the contents of the iterations, and can be quite challenging in a strictly time-boxed context