Agile/Lean & Safety: Perfect Match or Impossible Combination?

Mika Katara Matti Vuori
Department of Software Systems
Tampere University of Technology

This presentation reports results of the OHJELMATURVA project. Partial funding from Tekes and the companies participating in the project is gratefully acknowledged.

Safety Critical Software

- Trend towards using software to implement features that have been traditionally implemented in hardware
  - New complex features implemented in software
- Benefits of a software solution compared to hardware include: more advanced functionality without need to increase the size and 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

- Software safety is part of the overall safety of the system
- Software elements 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 – but software has to be robust
- The concept of reliability is fundamentally different for software and hardware: since the former does not wear, our confidence towards a software component increases if it has been in use for a long time without failures (“proven in use”)
- Software safety is assured by
  - Proper the development process: quality + safety assessment
  - Verification & Validation activities

IEC 61508-3 ed2.0

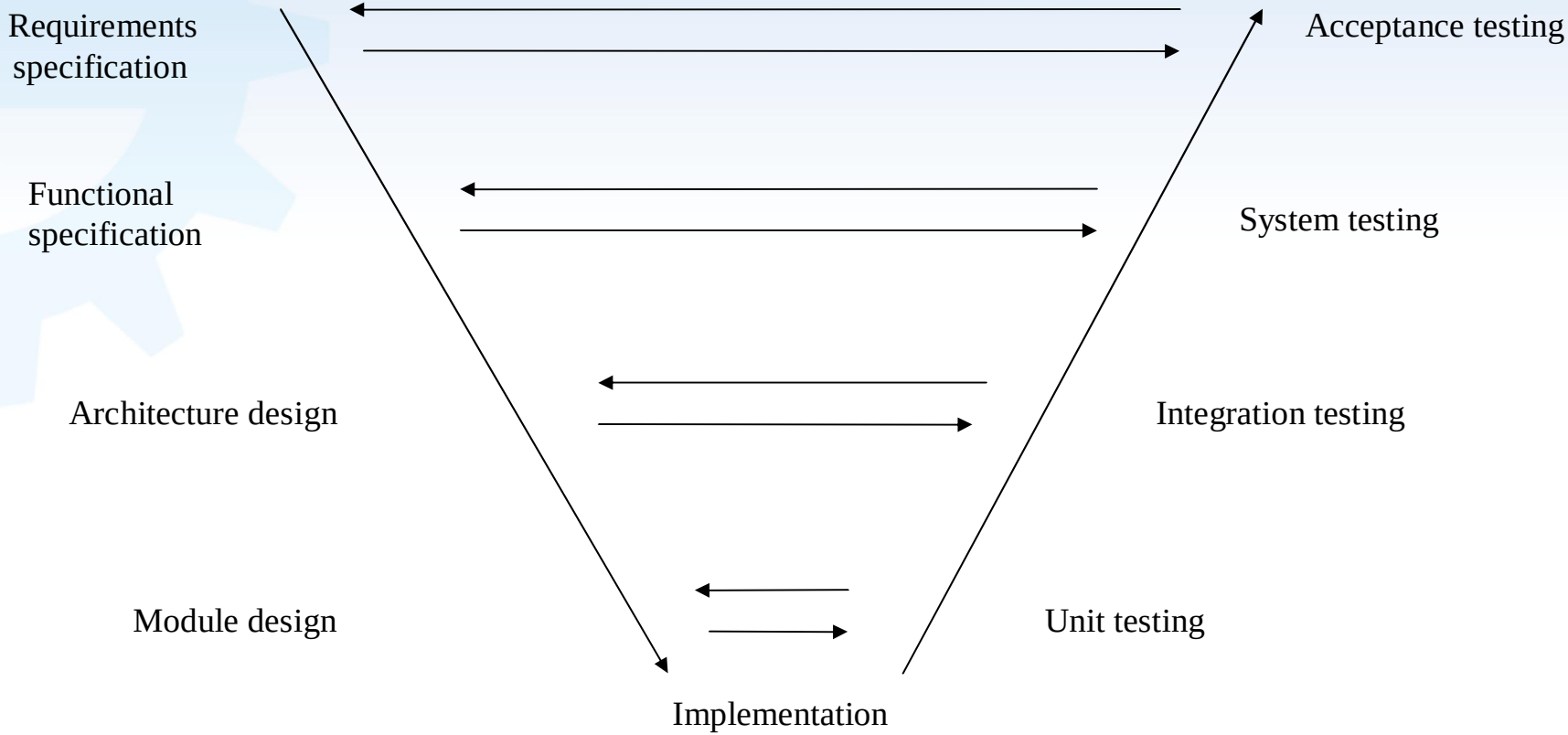
- While the lifecycle model of IEC 61508-3 ed2.0 is based on the traditional V-model type of process, some organizations are moving towards more agile and lean software development processes and practices
- Some organizations would like to tailor the development process to be more flexible, but still satisfy the standards' requirements
- Obviously, such tailoring must not compromise 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**
- **Moreover, while agile embraces change, do we really want to modify (certified) safety-related software?**

YES!
But it may require some effort

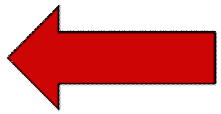
Table	Technique / Measure			Ref.	SIL 1	SIL 2	SIL 3	SIL 4
A.8.1	Impact Analysis			C.5.23	HR	HR	HR	HR
A.8.2	Reverify changed software module			C.5.23	HR	HR	HR	HR
A.8.3	Reverify affected software module			C.5.23	R	HR	HR	HR
A.8.4a	Revalidate complete system				—	R	HR	HR
	A.7.1	Probabilistic testing		C.5.1	—	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.5.5	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 Processes

- 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 Verification & Validation, 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 may be impossible to implement (due to schedule and/or budget constraints)
  - Often in practice, the requirements for the software part 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
- Iterative and incremental development reduces project risks compared to the pure V-model
- 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*



12 Agile Principles + thoughts on safety

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software. ***How to make sure that also the safety standards and legislation are satisfied?***
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage. ***How to make sure that the chance does not compromise safety?***
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale. ***Should all or most releases satisfy the safety requirements or only the "final" one (amount of testing needed manual/automatic)?***
4. Business people and developers must work together daily throughout the project. ***What is the role of safety engineers?***

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done***. In safety, trust is not enough – in most of the accidents caused by software, trust or lack of it has not been the issue.***
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation. ***Non-documented communication seldom produces proof that can be later examined to make sure that the right decisions have been made.***
7. Working software is the primary measure of progress. ***The term "working" should probably include "safe".***
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely. ***Many safety systems are embedded – sometimes the schedule of the hardware development poses restrictions on the software development.***

9. Continuous attention to technical excellence and good design enhances agility***. Safety needs also continuous attention.***
10. Simplicity–the art of maximizing the amount of work not done–is essential. ***Safety also requires simplicity so that the amount of V&V needed remains reasonable.***
11. The best architectures, requirements, and designs emerge from self-organizing teams. ***There needs to be a single person to sign the "Confirmation of Conformity", i.e. to take the responsibility. Who makes the difficult decisions? Who provides the second opinion, i.e. independent Verification & Validation?***
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly***. Learning is good, but safety needs to be ensured right from the beginning.***

Lean

- Originally called lean production, today represents a vision of good and efficient way of action
- Applied mostly in manufacturing industries, but making its way into software development
- Originates from Toyota's car manufacturing
  - 14 principles try to capture the essentials of the philosophy
- The Poppendiecks have been the most active in adapting lean into software production
  - Guiding principles: Eliminate Waste, Create Knowledge, Build Quality In, Defer Commitment, Optimize the Whole, Deliver Fast, Respect People
- Poppendiecks' principles are more agile than the original Toyota's principles – **lean is not the same as agile**

Lean vs. Agile (Coplien & Bjørnvig 2010)

Lean	Agile
Thinking and doing	*Doing*
Inspect-plan-do	Do-inspect plan
Feed-forward and feedback (*design* for change and respond to change)	Feedback (react to change)
High throughput	Low latency
Planning and responding	Reacting
Focus on process	Focus on people
Teams (working as unit)	Individuals (and interactions)
Complicated systems	Complex systems
Embrace standards	Inspect and adapt
Rework in design adds value, in making it is waste	*Minimize up-front work of any kind and rework code to get quality*
Bring decisions forward (Decision Structure Matrices)	Defer decisions (to the last responsible moment)

Lean Planning and Documentation

- Lean is sometimes misunderstood as mechanistic “waste hunt”
  - Documents and planning are discarded as artifacts and work that doesn’t produce value – **nothing could be further from the truth**
- Lean encourages planning to make things more efficient in execution and understands that documents are necessary to capture information, but the main use for them is to share knowledge between all people in the company, not just inside one team – thinking of the whole, and in the long term
- While lean manufacturing tries to maximize throughput by careful planning, what is the equivalent in software development?
  - Plans should be constantly updated as we learn new things about the system under development – changing requirements
  - Lean is plan-driven!

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, dividing responsibilities
  - 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 actual 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

Perfect Match or Impossible Combination?

- 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 (very much against the agile philosophy)

- 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 (regression testing & coverage)
  - Automatic document creation (if possible)
  - Requirements management (must)
  - Using tools to trace requirements (other than Excel)
  - Making reviews more effective (homeworkless reviews etc.)
- 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/lean 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 proven 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/leanness 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
  - Clear and modifiable code and architecture

- 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/lean requires 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

Steps Towards Productivity in Safety

- More thorough unit testing and code quality assurance so that there is a solid base for changing the product.
- Continuous integration so that there is always a working product at hand to assess & reflect.
- System-level test automation. Automating as much of any tasks that are required for verification and validation.
- Process automation – automated reporting tools and information systems that can easily be tailored to changing situations in a project.
- Automated tracing. When a requirement or design or implementation is being planned to change, has changed, tools are needed to immediately see the effects and what needs to be addressed.
- More self-reflection in the process. All participants need to look into the product and process frequently to see what in their way of action needs to be improved. This calls for frequent lessons learned / retrospect meetings in the process.
- Team building and improved leading of teams. Teams need to able to work as teams and learning that, and learning to lead teams in more collaborative ways, takes a time.
- Improved communications tools. Anything that helps people to communicate better, yet giving a peaceful environment to those tasks and persons that benefit from it.

Thank You

Contact:
Mika Katara
Department of Software Systems
Tampere University of Technology
mika.katara@tut.fi
P: +358 40 849 0743

Further reading:

Jani Paalijärvi : "Development of Safety-Critical Software using Agile Methods", Master of Science Thesis, Tampere University of Technology, May 2010.