Agile (sprints) and Safety

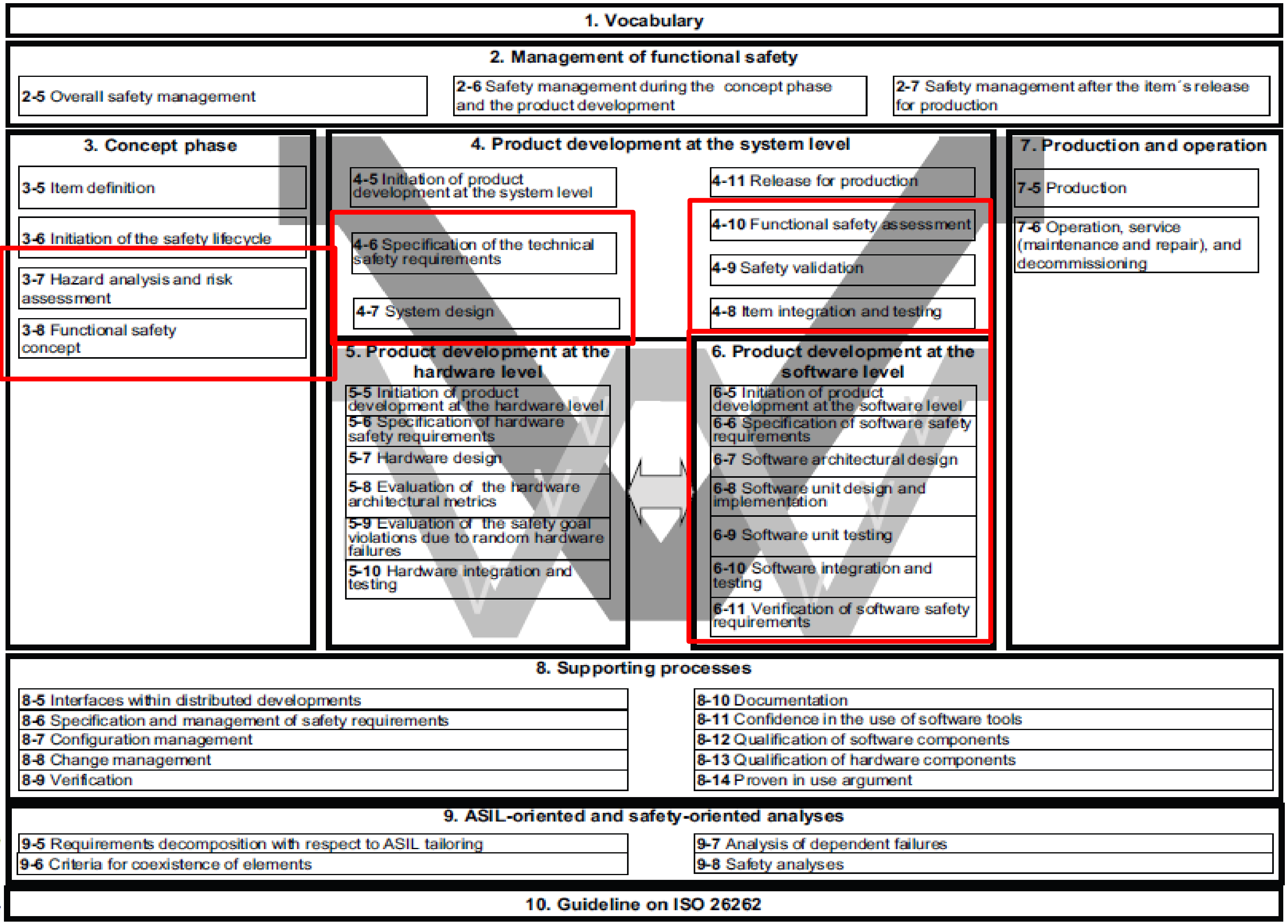
Even-André Karlsson

addalot
QUALITY IMPROVEMENT

Development and Agile and Safety

- All development
  - Goes from needs to solutions (”waterfall”)
  - Passes through some process, e.g.
    - ”requirements”,
    - ”design”
    - ”code”
    - ”test”
    - ”deliver”
- Safety requires quite a lot of ”overhead” in the development process
- Agile development is about dividing the ”functionality” into ”pieces”, that are so small and can be done so ”fast” that most of the steps above are ”obvious”
- It is also not obvious how we shall combine this with the Safety ”overhead”, e.g. do the Safety analysis and Safety Case in pieces

Safety activities of 26262 – High level



High level activities

Ensure that the system is safe; safety req => solution

- 3-7 Hazard analysis and risk assessment
- 3-8 Functional safety concept
- 4-6 Specification of the technical safety requirements
- 4-7 System design
- 4-8 Item integration and testing
- 4-9 Safety validation
- 4-10 Functional safety assessment
- 4-11 Release for production

Prove that you have done what is needed

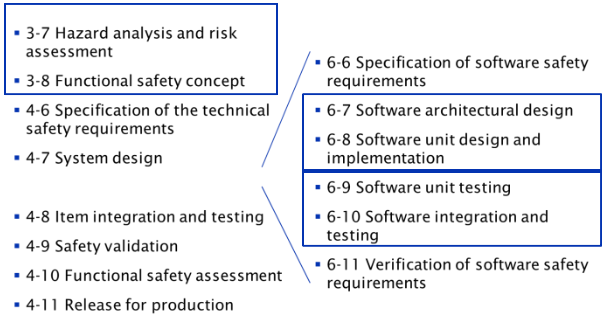
- 6-6 Specification of software safety requirements
- 6-7 Software architectural design
- 6-8 Software unit design and implementation
- 6-9 Software unit testing
- 6-10 Software integration and testing
- 6-11 Verification of software safety requirements

Ensure that no SW errors are introduced

addalot QUALITY IMPROVEMENT

Possible safety increments/sprints

1. Safety concept (20% into dev)
   => Safety requirements added to back-log
2. Defensive software design (50% into dev)
3. Safety test on sw level (60% into dev)
4. Safety verification and validation (70% into dev)
5. Review previous safety activities (80% into dev)
6. Final safety increment



Assuming 14 two weeks sprints = 28 weeks:

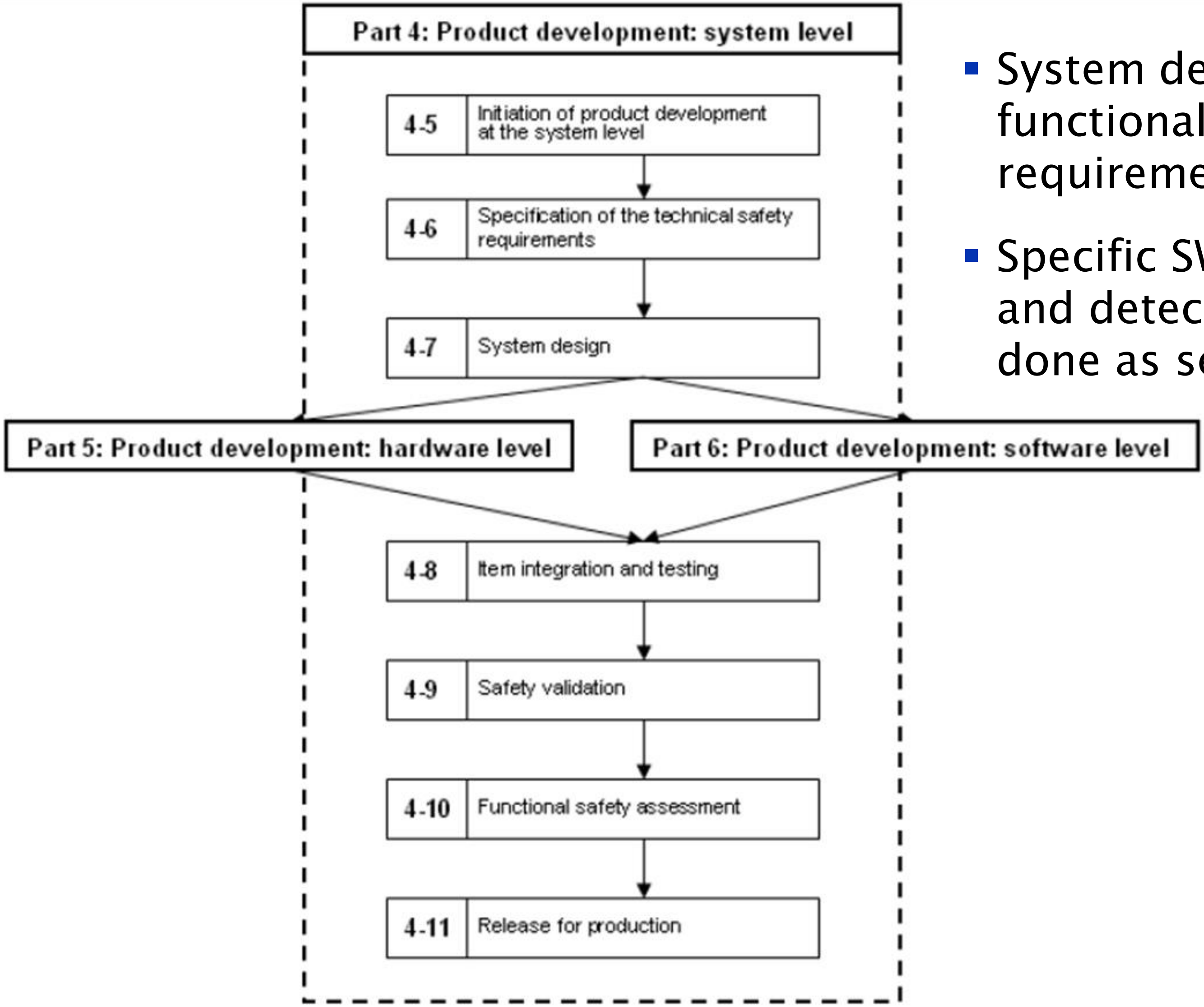
X X 1 X X X 2 X 3 X 4 5 X 6

6 of 14 sprints are safety focused. In addition there are safety requirements to be implemented

Concept activities

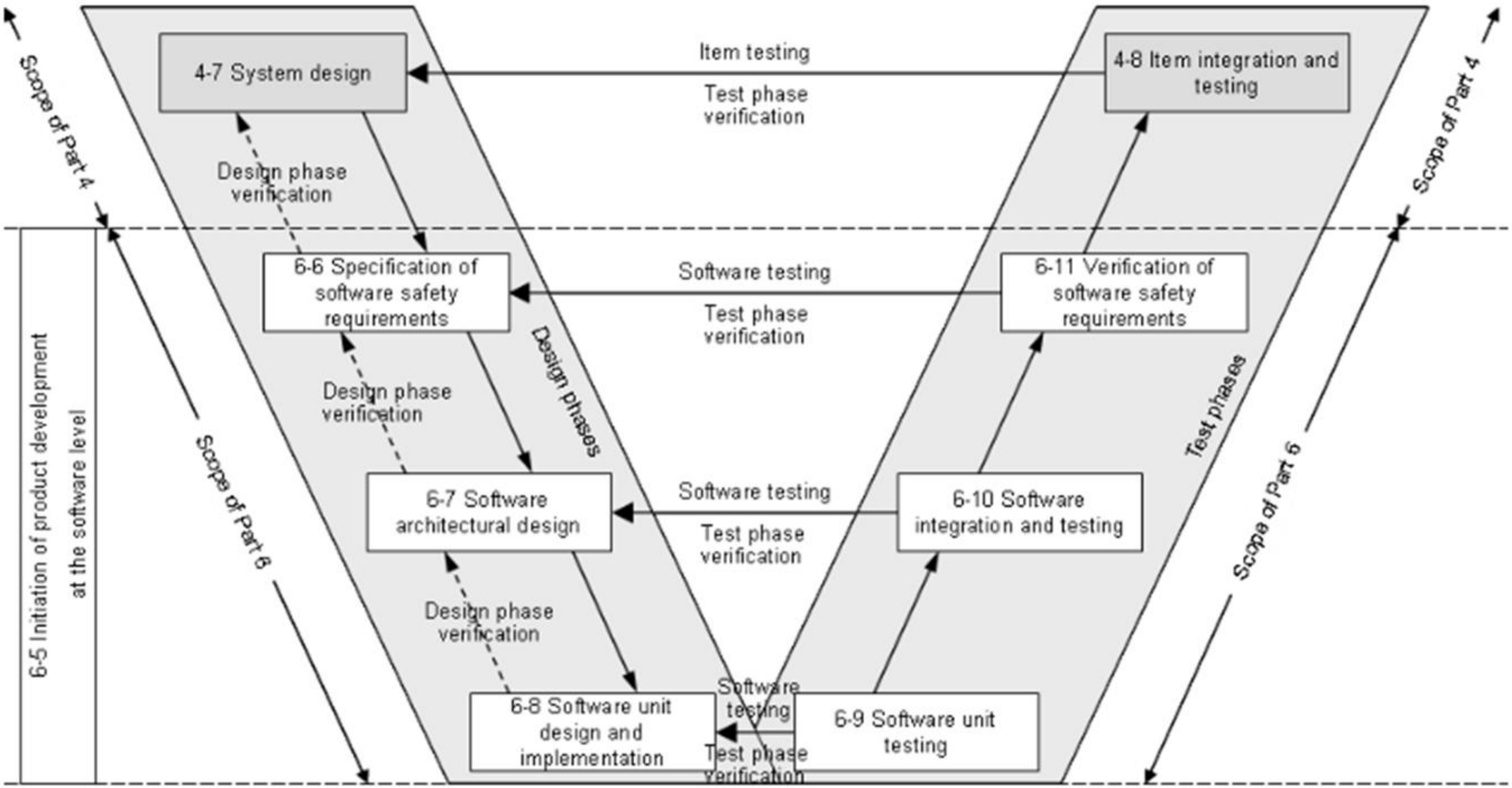
- 3.5 Item definition
  - We assume that enough is known about the "item" after the first two functional sprints. If this changes, we have to redo/complement some of the other activities.
- 3.6 Initiation of the safety lifecycle
  - It should already be clear if this is development from scratch or modifications. This presentation only discuss development from scratch, but could probably be applicable for modifications as well.
- 3.7 Hazard analysis and risk assessment
  - One of the main activities of the first safety sprint
    - 7.4.2 Situation analysis and hazard identification
    - 7.4.3 Classification of hazardous events (severity, probability, controllability => ASIL)
    - 7.4.4 Determination of ASIL and safety goals
    - 7.4.5 Verification
- 3.8 Functional safety concept
  - Another main activity of the first safety sprint
    - This results in a set of "functional" safety requirements that can be implemented in sprints

System design activities



- System design is done per functional safety requirement in sprints
- Specific SW error prevention and detection activities are done as separate sprints

SW design activities



addalot QUALITY IMPROVEMENT

Detailed SW design activities

- 6-6: Specification of software safety requirements
  => Part of each functional sprint
- 6-7: Software architectural design
  => Activities split between functional sprints and defensive design sprint
- 6-8: Software unit design and implementation
  - 6-8.4.2 Design Notation
  - 6-8.4.3 Design
  - 6-8.4.4 Design principles
  - 6-8.4.5 Verification

  => Activities done as part of functional sprint => More documentation
- 6-9: Software unit testing
  => Testing split between functional sprints and specific safety test sprint
- 6-10: Software integration and testing
  => Testing split between functional sprints and specific safety test sprint

addalot QUALITY IMPROVEMENT

Defensive software design increment (1)

Table 4 — Mechanisms for error detection at the software architectural level

Methods		ASIL A	ASIL B	ASIL C	ASIL D
1a	Range checks of input and output data	++	++	++	++
1b	Plausibility check[a]	+	+	+	++
1c	Detection of data errors[b]	+	+	+	+
1d	External monitoring facility[c]	o	+	+	++
1e	Control flow monitoring	o	+	++	++
1f	Diverse software design	o	o	+	++

[a] Plausibility checks can include using a reference model of the desired behaviour, assertion checks, or comparing signals from different sources.

[b] Types of methods that may be used to detect data errors include error detecting codes and multiple data storage.

[c] An external monitoring facility can be for example an ASIC or another software element performing a watchdog function.

Note 1f will require much more work, and has to be planned up front as separate sprints

addalot QUALITY IMPROVEMENT

Defensive software design increment (2)

Table 5 — Mechanisms for error handling at the software architectural level

Methods		ASIL A	ASIL B	ASIL C	ASIL D
1a	Static recovery mechanism[a]	+	+	+	+
1b	Graceful degradation[b]	+	+	++	++
1c	Independent parallel redundancy[c]	o	o	+	++
1d	Correcting codes for data	+	+	+	+

[a] Static recovery mechanisms can include the use of recovery blocks, backward recovery, forward recovery and recovery through repetition.

[b] Graceful degradation at the software level refers to prioritizing functions to minimize the adverse effects of potential failures on functional safety.

[c] Independent parallel redundancy can be realized as dissimilar software in each parallel path

addalot QUALITY IMPROVEMENT

Software safety test increment

Table 10 — Methods for software unit testing

	Methods	ASIL A	ASIL B	ASIL C	ASIL D
1a	Requirements-based test[a]	++	++	++	++
1b	Interface test	++	++	++	++
1c	Fault injection test[b]	+	+	+	++
1d	Resource usage test[c]	+	+	+	++
1e	Back-to-back comparison test between model and code, if applicable[d]	+	+	++	++

Table 11 — Methods for deriving test cases for software unit testing

	Methods	ASIL A	ASIL B	ASIL C	ASIL D
1a	Analysis of requirements	++	++	++	++
1b	Generation and analysis of equivalence classes[a]	+	++	++	++
1c	Analysis of boundary values[b]	+	++	++	++
1d	Error guessing[c]	+	+	+	+

Conclusion

Each sprint is completed with defined exit criteria

1. Hazard analysis to determine functional safety requirements
   => Hazard analysis and functional safety requirements added to backlog
2. Functional safety requirements are implemented in sprints with ”normal testing”
   => Normal design, but with required safety documentation
3. ”Defensive software design” is done in separate sprint
   => Analysis and documentation of ”defensive software design”
4. Additional safety testing
   => Analysis and documentation of ”safety testing”
5. Safety validation is summarized in last increment
   => Safety case

addalot QUALITY IMPROVEMENT

Advantages of an incremental approach

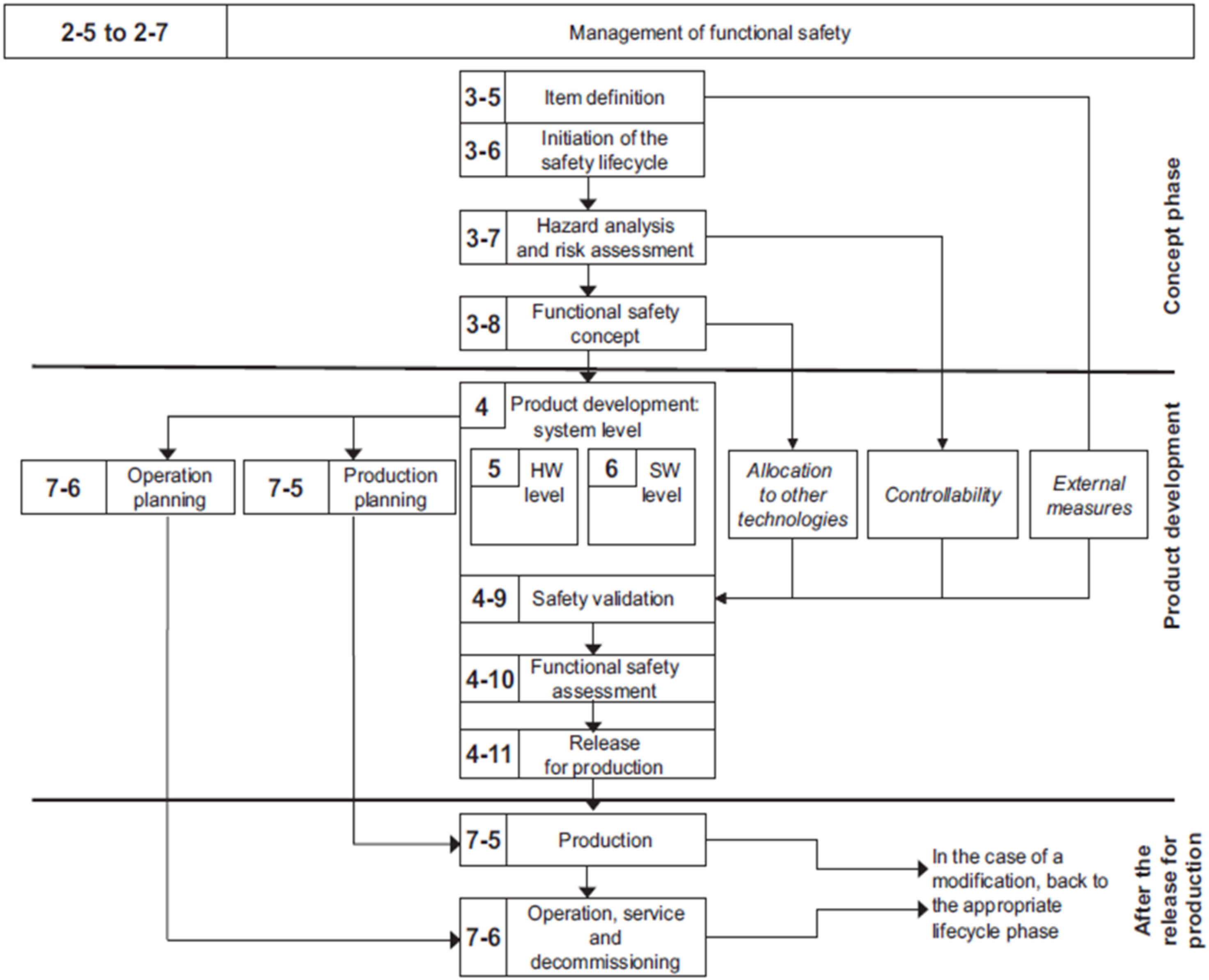
- Focus the whole team on safety activities
- Able to bring in experts to help out
- Ensure that the safety activities are done
- Implementing the functional safety requirements in sprints will improve the ”automatic” traceability provided by the versioning of artefacts.

"Excellent firms don't believe in excellence - only in constant improvement and change."

In Search of Excellence - Tom Peters

Nicolas.Martin-Vivaldi@addalot.se
+46 706 800 521

addalot
QUALITY IMPROVEMENT


2-5 to 2-7
Management of functional safety
3-5
Item definition
3-6
Initiation of the safety lifecycle
3-7
Hazard analysis and risk assessment
3-8
Functional safety concept
Concept phase
4
Product development: system level
5
HW level
6
SW level
7-6
Operation planning
7-5
Production planning
Allocation to other technologies
Controllability
External measures
4-9
Safety validation
4-10
Functional safety assessment
4-11
Release for production
Product development
7-5
Production
7-6
Operation, service and decommissioning
In the case of a modification, back to the appropriate lifecycle phase
After the release for production