Traceability in an Agile process

Even-André Karlsson

addalot
QUALITY IMPROVEMENT

Content

- Background
- Connections to Safety
- Approach
- Technicalities
  - Requirements structure
  - Architecture and design
  - Code and unit test
  - Test cases
  - Change requests
- Experiences
- Tomorrow's workshop:
  - Examples
  - Special cases
  - Detailed discussion on fulfillment of ASPICE requirements

Background

Customer

- wanted to achieve ASPICE compliance
- had already a working agile process

Can we implement traceability in an agile way?

Automotive SPICE has very high requirements on traceability:

An overview of bidirectional traceability and consistency is depicted in the following figure.

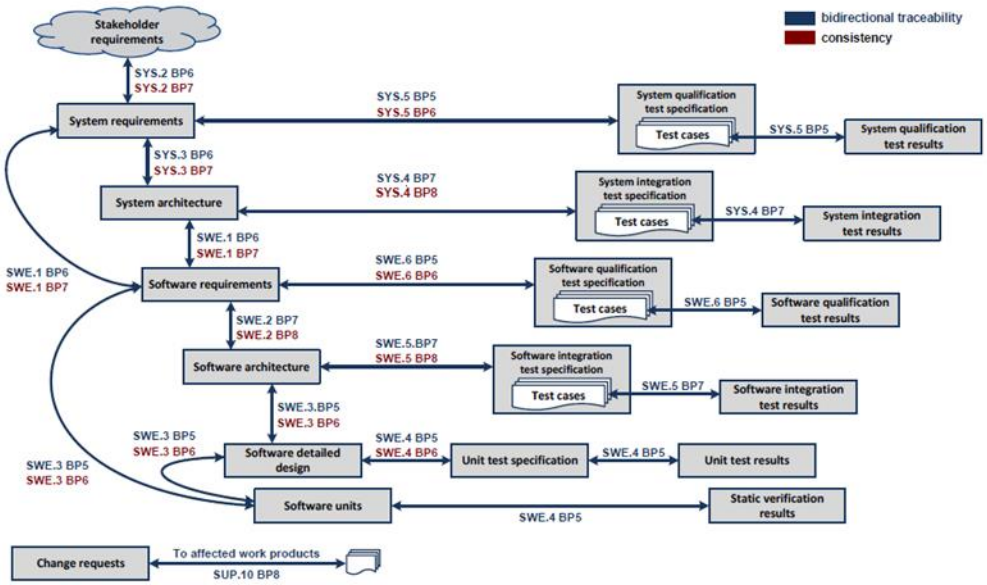


Figure D.4 — Bidirectional Traceability and Consistency

addalot QUALITY IMPROVEMENT

Connections to Safety

- The Safety Standards also require traceability, example from 26262 (5)

> **7.4.2** During the development of the software architectural design the following shall be considered:
>
> a) the verifiability of the software architectural design;
>
> NOTE This implies bi-directional traceability between the software architectural design and the software safety requirements.
>
> b) the suitability for configurable software;

8.4.5 The software unit design and implementation shall be verified in accordance with ISO 26262-8:2011 Clause 9, and by applying the verification methods listed in Table 9, to demonstrate:

a) the compliance with the hardware-software interface specification (in accordance with ISO 26262-5:2011, 6.4.10);

b) the fulfilment of the software safety requirements as allocated to the software units (in accordance with 7.4.9) through traceability;

addalot QUALITY IMPROVEMENT

Connection to Safety (2)

- 26262 – Section 8

Figure 2 — Structure of safety requirements

The management of safety requirements includes managing requirements, obtaining agreement on the requirements, obtaining commitments from those implementing the requirements, and maintaining traceability.

6.4.3.2 Safety requirements shall be traceable with a reference being made to:

a) each source of a safety requirement at the upper hierarchical level,

b) each derived safety requirement at a lower hierarchical level, or to its realisation in the design, and

c) the specification of verification in accordance with 9.4.2.

NOTE Additionally, traceability supports:

— an impact analysis if changes are made to particular safety requirements, and

— the functional safety assessment.

8.4.1 Planning and initiating change management

8.4.1.1 The change management process shall be planned and initiated, before changes are made to work products.

NOTE Configuration management and change management are initiated at the same time. Interfaces between the two processes are defined and maintained to enable the traceability of changes.

General idea

1. Requirements are hierarchically structured in
   - Personas
     - Journeys
       - Epics
         - Stories
2. Stories are broken down into tasks during sprint planning
3. Each task is implemented separately
4. We are versioning all our artifacts, so by using the Task ID as a check in comment we will get some traceability

Can we use this, and will this be enough?

Will it be bidirectional?

Development context

- No advanced tools: Word, Excel, Code
- Good CM systems both for documents (Alfresco) and code (SVN)
- Only Software, i.e. No System/Software level
- Functional testing is only done on total System/Software level per sprint
- Reasonably small products, i.e. teams of 6-8 people, but complete responsiblity

Requirements and planning

- Requirements structure
  - Personas
    - Journeys
      - Epics
        - Stories
          - Tasks (these are implemented)

One Excel document that is kept updated

Product Backlog

One Word document per sprint

Sprint planning

- The Product backlog is maintained throughout the life time of the product
- When we start a sprint we record which Stories are part of this sprint, i.e. add (R1.S4 = Release 1 Sprint 4) in the implementation column.
  - Note that a story can be implemented over several Sprints, then we have several references.
  - Check in the Product Backlog in Alfresco again with Commit comment “Sprint R1.S4 planning”.
  - Updated at the end of the sprint if needed.

Detailed cases

- ”System” requirements
  - Added a new Persona: The Architect
- Changing stories and tasks in later releases: CR part
- Handling versions in Excel : CR part
- TaskID format and which tasks to track
  - Only coding tasks are used for tracking, even if test and documentation tasks are used in planning

Architecture

- Tasks requiring architecture updates are identified during sprint planning
  - Not all stories and tasks have architectural impacts
- Changes are done per task to the Word architecture document

  Keep track changes always on (except when updating Word admin content)
  1. Check out
  2. Accept all previous changes
  3. Do the technical changes
  4. Use Word Index function every time a unit is mentioned
     - Index is used for backwards traceability from components to architecture
     - Also index components used in diagrams, e.g. Object, Sequence…
     - All components shall be mentioned in architecture
  5. Check in with Task ID in comment

This solves traceability from requirements to architecture

What about from architecture to requirements? No blame function in Word ☹

- We have decided to do this indirectly:
  - Details and example for tomorrow

addalot QUALITY IMPROVEMENT

Architecture

- Traceability between architecture and code
  - All modules are described in architecture
  - Use of Index ensures that we have bidirectional traceability
  - Modules used in pictures must be explicitly indexed (if not mentioned in the text explaining the component (then they are indexed in the text).
- What to minimally cover in the architecture document?
  - Already specified in ASPICE
  - When we add the functionality we update the architecture
  - Can of course have design/spike documents on the side before we introduce it “for real”
- Traceability from architecture back to requirements
  - We do not have any “blame” function in Word, so we can not point to a piece of text and ask where it come from ☹ Can quite easily be implemented….
  - But we can find all changes to a module in the module (task)
  - Then we can find which of these had architecture impact, and find the changes
  - Non functional aspects of the architecture are more complex. One approach here is explicitly add tasks impacting these chapters in the architecture document (not done yet)
- Deep architectural changes
  - Do the changes in a separate document, and check it in with the or new functional task ID’s

Code

- Code and unit test cases are in SVN
- Changes are done per task and checked in with task ID
- To find code impacted by a requirement is just to search the commit comments
- We can use the blame function to find the latest check in that changed any line, i.e. the task => the requirement
- If we go to that version in we can go further back in time

Code

- Code refactoring
  - Simple, e.g. add a parameter to a function
    1. For a new function: Use the new function task, but add a note in the check-in comment that this is an addition, so we know we have to go further back in history if we want the full story
       (the blame function could be extended to find all check in's impacting a line)
    2. If an improvement of the original functionality: use the old task id. That is easily found in by the blame.
  - Complex, i.e. destructive
    - Find the functionalities originally requiring the code (blame). This is a good exercise to understand what we need to retest.
    - Do the restructuring outside
    - Check in the new code in pieces with the old task ID's (note restructuring in the commit comment)
- Splitting files
  - Use the copy file function in SVN, that will also copy the history
  - Remove the superfluous parts from each file
- Handling defects
  - When you have found the line to change (if you know which functionality it is related to the Task ID can help you find it.)
  - Find the original task ID that created the faulty code (blame)
    - It can be that the task modifying the code has nothing to do with the faulty functionality, i.e. we have assumed something that did not work as we thought, or someone has changed something that we relied on
  - Check in the change with the correct task ID

Test cases

- Test cases are written in Excel and executed per sprint
- We mark explicitly which task it comes from in a column
- Test results are recorded in the same Excel sheet
- Sprint test cases are moved to a total test sheet
- For regression test sprint test cases are selected from the total test sheet to the sprint test sheet.
- In the main sheet we keep one column per sprint to get an overview of what we have tested when

Test cases

- Updating test cases
  - Since test cases are local to sprints, we will just copy and update the test case for a new sprint.
  - In the overall test case listing we will keep two versions of the test case, each belonging to it's own sprint(s).
- Examples: see next slide

Change requests

- Change requests are implemented as changes to the requirements
- A new version of the requirement document is checked in with new or updated Stories
- The CR is explicitly recorded in a column
- The change request ID(s) is used as the check in comment
- New stories are no problem, as their implementation will be traces as any other not yet implemented story
- Updated stories: For workshop

Change requests: Updated stories

- New version of the story is created in the Excel sheet (new line, as the old story is still implemented in the previous sprint/release).
- Two cases:
  - Just added tasks:
    - The story is reintroduced in a new sprint planning document with both new and old version, and explaining the difference.
    - New tasks are implemented. In the check in comment we need to add that these belong to the new version of the story
  - Changed tasks. Similar as above, but we reuse the old task ID. Check in's area marked that they belong to the new version of the task.
    - Note that both old (still valid as well as overwritten) and new check-ins related to task will come up when we search for the task ID
  - Update the old sprint planning document with information that the story/tasks are updated in the new sprint planning.
- Note: A CR is "Completed" when the Release Planning document is updated. We will then trace the change through the new or changed stories or tasks.

Experience

- Initially quite skeptical – will be a lot of work ☹
- Turned out to be quite simple – need to be meticulous, but each step is quite simple and requires little extra effort
- Also helpful in the normal work, i.e. not only overhead ☺
  - We can see why we changed things
  - Good to select regression test cases
  - SVN blame function very useful
  - Would be good with similar support in Word!
- Still open:
  - Will we be able to handle large restructurings?
  - Will this be practical after 5 years?

An overview of bidirectional traceability and consistency is depicted in the following figure.

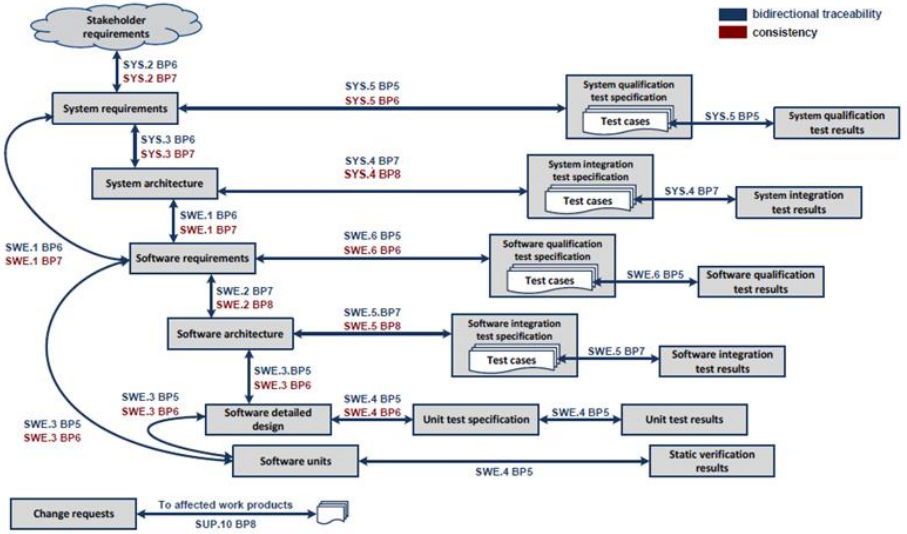
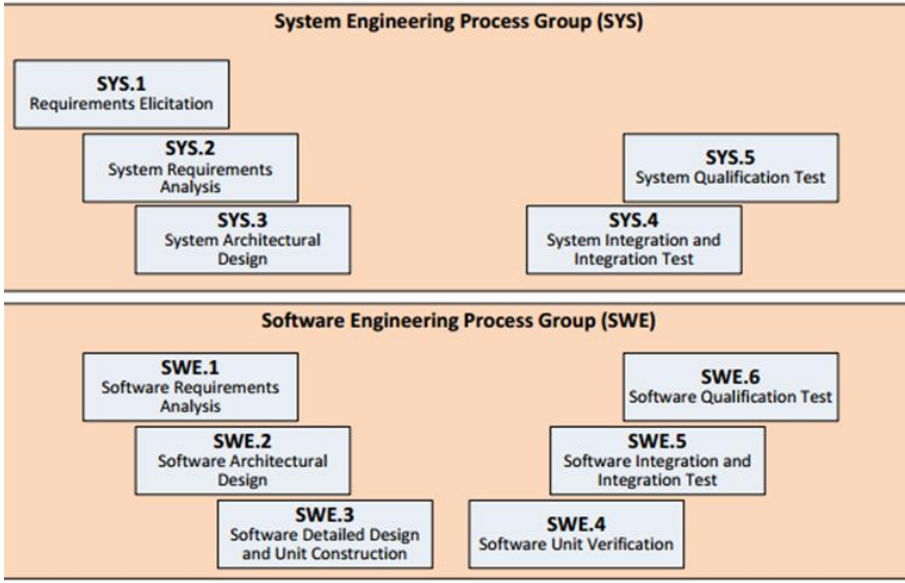


Figure D.4 — Bidirectional Traceability and Consistency

Comparing to overall ASPICE model

- ASPICE assumes a system level and SW level of both requirements and architecture – we only have a hierarchical breakdown
- ASPICE assumes Detailed design and Unit Construction – we have arch and doxygen and code
- ASPICE assumes different verification steps:
  - Unit test - OK
  - Software Integration test
  - Software Qualification test
  - System Integration test
  - System Qualification test

Just one level



addalot QUALITY IMPROVEMENT

ASPICE traceability

Weak links, all because of lack of blame function in Word ☹
Possible solution: Store pure text version in SVN (automatically) for each check in ☺

An overview of bidirectional traceability and consistency is depicted in the following figure.

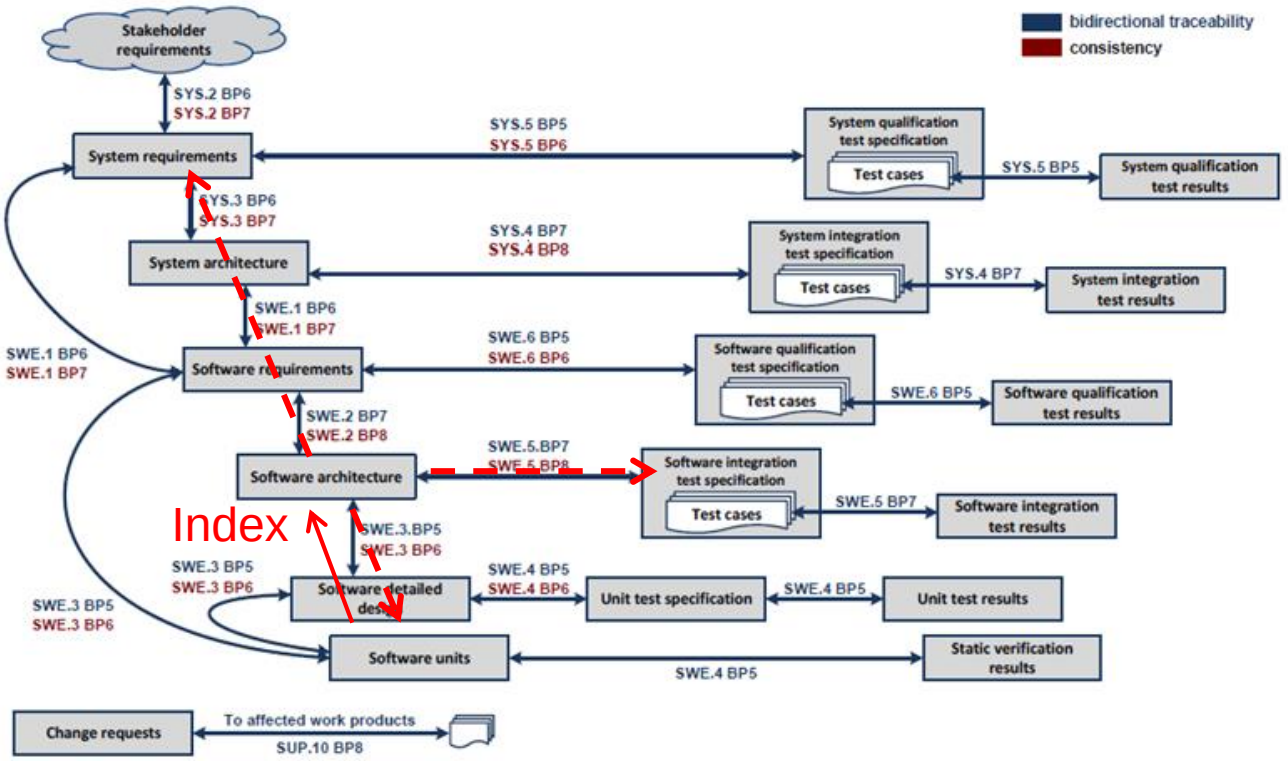


Figure D.4 — Bidirectional Traceability and Consistency

addalot QUALITY IMPROVEMENT

Ideal blame - historical

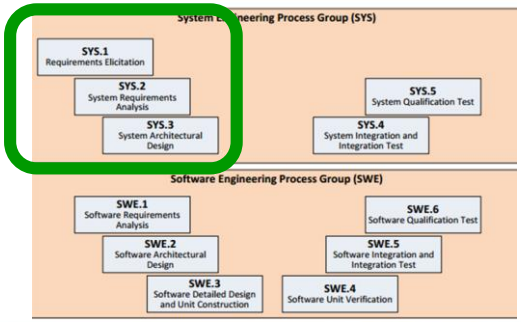
- When hoovering over a line we would like to see the whole history, e.g.:

Check-in 4	Line content
Check-in 8	Line content with change1
Check-in 12	Line content with change1 and change2
Check-in 20	Line content with change1, change3 and change2
Check-in 23	Line content with change1, update4 ~~change3~~ and change2

- Red text is new, Blue is unchanged and ~~strikethrough~~ is removed

Compared to ASPICE requirements, SYS1-3

- **Requirements Elicitation SYS.1.BP1: Obtain stakeholder requirements and requests.**
  - *NOTE 3: The information needed to keep traceability for each customer requirement has to be gathered and documented.*
- **System Requirements Analysis SYS.2.BP6: Establish bidirectional traceability.** Establish bidirectional traceability between stakeholder requirements and system requirements. [OUTCOME 6]
- **System Architectural Design SYS.3.BP6: Establish bidirectional traceability.** Establish bidirectional traceability between system requirements and elements of the system architectural design. [OUTCOME 5]
  - *NOTE 4: Bidirectional traceability covers allocation of system requirements to the elements of the system architectural design.*
  - *NOTE 5: Bidirectional traceability supports coverage, consistency and impact analysis.*



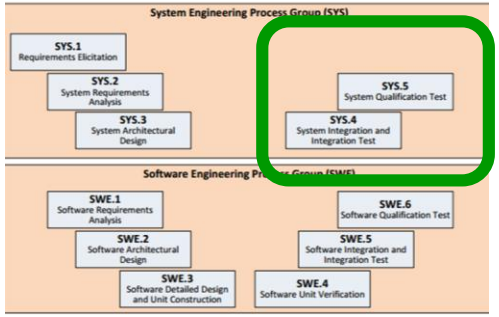
addalot QUALITY IMPROVEMENT

Questions/comments, SYS1-3

- Do we have a System and SW level?
- Can/must we map persona and journeys to customers?
- Initial architecture
  - When we create the architecture we usually make some large effort in the beginning, covering a lot of functionality.
  - Not so clearly connected to requirements.
  - Should be included in the “real” document as it is implemented to ensure traceability.
  - If necessary introduce system stories. Good exercise to motivate design.

Compared to ASPICE requirements, SYS4-5

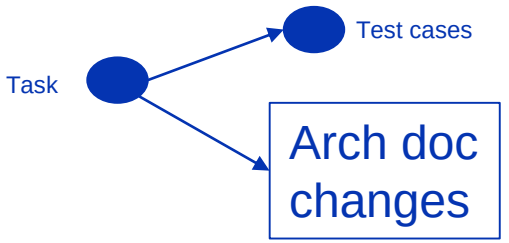
- **System Integration and Integration Test SYS.4.BP7: Establish bidirectional traceability.** Establish bidirectional traceability between elements of the system architectural design and test cases included in the system integration test specification.
  - Establish bidirectional traceability between test cases included in the system integration test specification and system integration test results. [OUTCOME 7]
  - *NOTE 7: Bidirectional traceability supports coverage, consistency and impact analysis.*
- **System Qualification Test SYS.5.BP5: Establish bidirectional traceability.** Establish bidirectional traceability between system requirements and test cases included in the system qualification test specification. Establish bidirectional traceability between test cases included in the system qualification test specification and system qualification test results. [OUTCOME 5]
  - *NOTE 2: Bidirectional traceability supports coverage, consistency and impact analysis.*



addalot QUALITY IMPROVEMENT

Questions/comments, SYS4-5

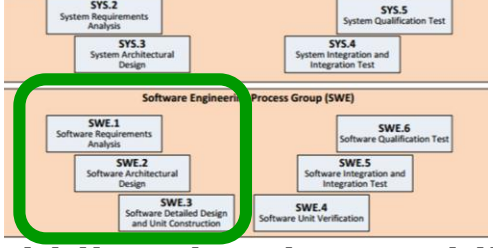
- What different “levels/types” of tests do we have?
- Will the indirect linking of test cases to tasks to changes in architecture be enough?
  - Note that if there are many test cases and changes connected to a task, the granularity of traceability will be low.
  - Can be controlled by splitting tasks



- Requires traceability from architecture to test cases
- Requires traceability to test results: OK since in same excel file.

addalot QUALITY IMPROVEMENT

Compared to ASPICE requirements, SWE1-3



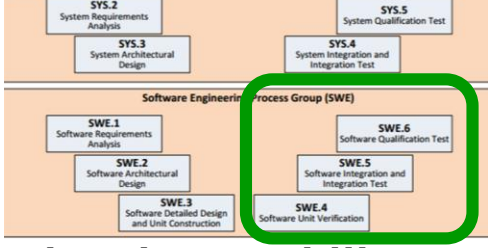
- **Software Requirements Analysis SWE.1.BP6: Establish bidirectional traceability.** Establish bidirectional traceability between system requirements and software requirements. Establish bidirectional traceability between the system architecture and software requirements. [OUTCOME 6]
  - *NOTE 8: Bidirectional traceability supports coverage, consistency and impact analysis.*
- **Software Architectural Design SWE.2.BP7: Establish bidirectional traceability.** Establish bidirectional traceability between software requirements and elements of the software architectural design. [OUTCOME 5]
  - *NOTE 6: Bidirectional traceability covers allocation of software requirements to the elements of the software architectural design.*
  - *NOTE 7: Bidirectional traceability supports coverage, consistency and impact analysis.*
- **Software Detailed Design and Unit Construction SWE.3.BP5: Establish bidirectional traceability.** Establish bidirectional traceability between software requirements and software units. Establish bidirectional traceability between the software architectural design and the software detailed design. Establish bidirectional traceability between the software detailed design and software units. [OUTCOME 4]
  - *NOTE 3: Redundancy should be avoided by establishing a combination of these approaches that covers the project and the organizational needs.*
  - *NOTE 4: Bidirectional traceability supports coverage, consistency and impact analysis.*

addalot QUALITY IMPROVEMENT

Questions/comments, SWE1-3

- Same questions as for System architecture if a large architecture document is established covering a lot of stories/tasks. How to achieve traceability?
- Do we need SW detailed design, or can we use Doxygen?
- Can we implement traceability from SW architecture design to low level design by making an index of all “modules” in the Architecture document?

Compared to ASPICE requirements, SWE4-6



- **Software Unit Verification SWE.4.BP5: Establish bidirectional traceability.** Establish bidirectional traceability between software units and static verification results. Establish bidirectional traceability between the software detailed design and the unit test specification. Establish bidirectional traceability between the unit test specification and unit test results. [OUTCOME 4]
  - *NOTE 7: Bidirectional traceability supports coverage, consistency and impact analysis.*
- **Software Integration and Integration Test SWE.5.BP7: Establish bidirectional traceability.** Establish bidirectional traceability between elements of the software architectural design and test cases included in the software integration test specification. Establish bidirectional traceability between test cases included in the software integration test specification and software integration test results. [OUTCOME 7]
  - *NOTE 6: Bidirectional traceability supports coverage, consistency and impact analysis.*
- **Software Qualification Test SWE.6.BP5: Establish bidirectional traceability.** Establish bidirectional traceability between software requirements and test cases included in the software qualification test specification. Establish bidirectional traceability between test cases included in the software qualification test specification and software qualification test results. [OUTCOME 5]
  - *NOTE 2: Bidirectional traceability supports coverage, consistency and impact analysis.*

addalot QUALITY IMPROVEMENT

Questions/comments, SWE4-6

- Requires both traceability from
  - Requirements to test cases and
  - Design to test cases

  On all levels of design and test

  Traceability from Test cases to Test results, i.e. we need to know which test cases have been run, when and the result.

Some references

Quite well discussed on the web, e.g.:

- http://pagilista.blogspot.be/2012/07/requirements-traceability-in-agile.html
- https://www.rallydev.com/blog/agile/high-assurance-agile-software-development-traceability-matrix-examples
- http://fileadmin.cs.lth.se/cs/education/examensarbete/rapporter/2009/2009-02_rapport.pdf

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

In Search of Excellence - Tom Peters

Even-Andre.Karlsson@addalot.se
+46 706 800 533

addalot
QUALITY IMPROVEMENT