



Traceability for Impact Analysis and Co-Evolution

Luc Boillat, Ivo Indergand

Group 3



Software maintenance

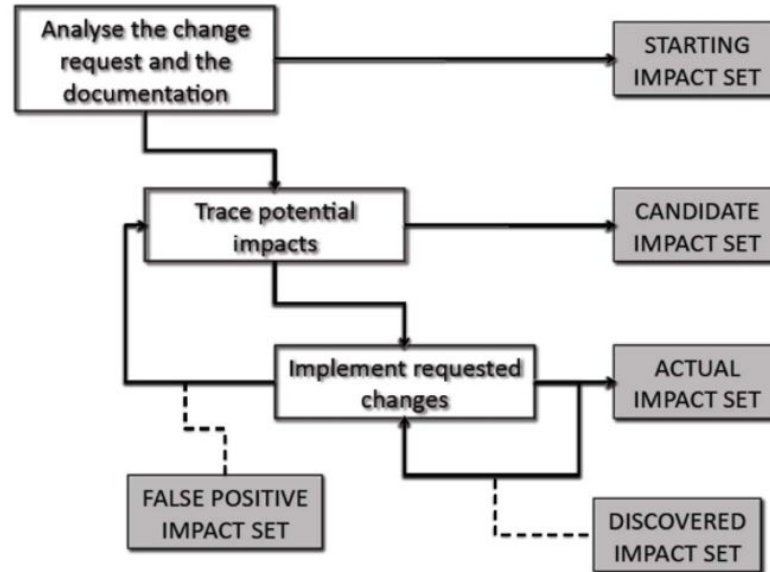
- Definition: *The process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment*
- Classified into adaptive, perfective and corrective maintenance activities
- Changes have effect on artefacts like source code, requirements, designs and test artefacts



Why apply traceability

- Identify ripple effects
- Allow consideration of trade-offs between suggested software change approaches

Impact analysis process





Approaches

Framework dimensions:

- Vertical- or horizontal traceability links
- Source of information used to derive them (structural or based on knowledge)
- Its implicit- or explicit representation with the traceability management system



Challenges

- Recovery due to heuristics-, information retrieval- or data-mining
- Link evolution



Supporting requirements update during software evolution

- Eya Ben Charrada, Anne Koziolk and Martin Glinz 2015

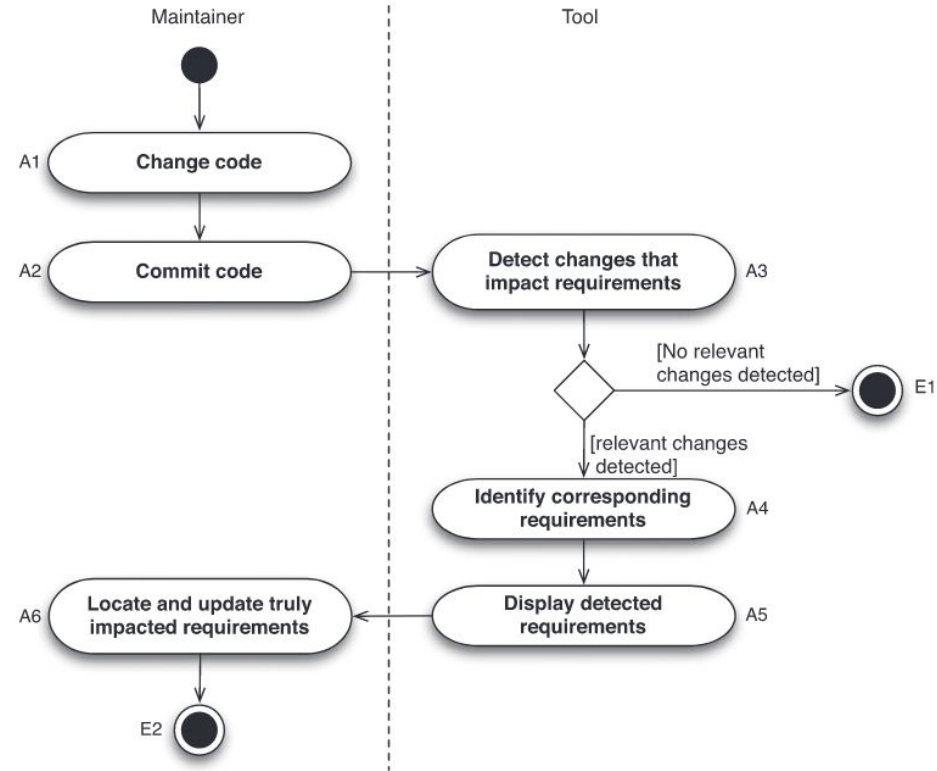


Motivation of Ben Charrada 2015

- In software projects requirements specification often becomes obsolete since requirements do not get updated
- Requirements update is a manual task that is time consuming
- An automated approach would support this task
- Detect outdated requirements that are impacted by software change

General Procedure

- After commit, press button to see ranked list of potentially impacted requirements
- Differencing Step
- Tracing Step
- Displaying Step
- Much faster than manual review





Identifying Relevant Changes (External Behavior)

- Addition of new elements
- Changes in private elements

Relevant

- Changes in method bodies
- Rename operations
- Changes in method signatures

Irrelevant



Tracing Changes Back to Requirements

- Extracting keywords from changed elements, their documentation and their call hierarchy
- Group keywords by class into single files to not lose specificity to the changes
- Compare lists to requirements specification using IR-based tracing techniques
- Results in a ranked requirements list for each class
- Merge the lists into one ranked list, based on the requirements' rank in the class lists and their frequency across all lists



Case Study

- Tested on 3 software projects
- 2 Research questions
 - How good is the approach in **identifying the commits that impact requirements** and in filtering out those that do not impact requirements?
 - Measured in True Positive Rate, True Negative Rate and Accuracy
 - For relevant code changes, that is, those that impact requirements, how good is the approach in **identifying the requirements that are impacted?**
 - Measured in Recall, Precision and Fall Out

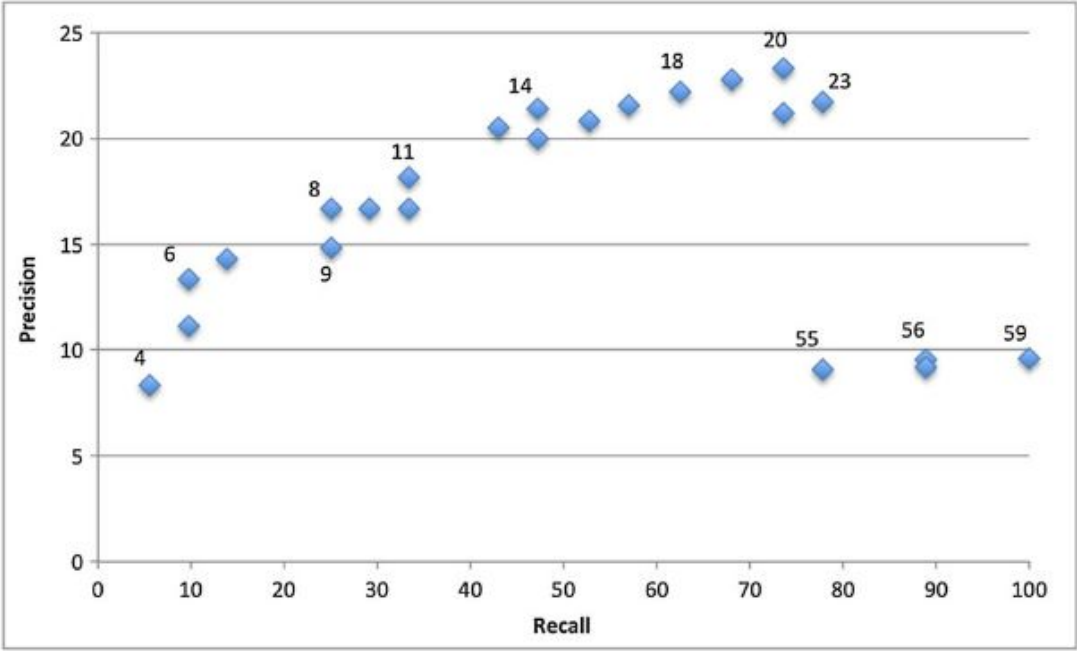


“AquaLush” Case Study Results

- Irrigation system that was developed as a benchmark for traceability
- Ground Truth was identified manually as a reference to the automated approach
- Accuracy 88.8%
- True Positive Rate 100%
- True Negative Rate 83.3%
- Recall 73.6%
- Precision 23.3%
- Fall-out 4.6%

Rank	Change 1	Change 2	Change 3
1	SRS383	SRS358	SRS237
2	SRS33	SRS24	SRS24
3	SRS384	SRS367	SRS97
4	SRS254	SRS28	SRS53
5	SRS53	SRS366	SRS293
6	SRS30	SRS253	SRS69
7	SRS378	SRS368	SRS252
8	SRS31	SRS369	SRS238
9	SRS52	SRS377	SRS277
10	SRS32	SRS27	SRS52
11	SRS379	SRS97	SRS98
12	SRS145	SRS376	SRS291
13	SRS29	SRS25	SRS48
14	SRS186	SRS360	SRS240
15	SRS24	SRS187	SRS112
16	SRS48	SRS26	SRS32
17	SRS153	SRS374	SRS109
18	SRS69	SRS359	SRS333
19	SRS380	SRS53	SRS110
20	SRS98	SRS69	SRS150
21	SRS47	SRS52	SRS289
22	SRS97	SRS113	SRS27
23	SRS382	SRS167	SRS99
24	SRS49	SRS277	SRS96
25	SRS381	SRS109	SRS269
Missed	0	0	2

Outdated
Related





Evaluation

- Generally good results
- Even results with medium precision save the maintainer time, as it's still faster than reviewing all requirements manually and idea was to support, not replace human action



Questions