



Traceability and Requirements Evolution

Group 2 - Andreas Schaufelbühl, Sacha Uhlmann

Agenda

— — —

Motivation

Tracing and Impact Analysis

Tracing as Support for Requirement Updates in Software Evolution

Discussion

Motivation

— — —

Mandatory Readings discuss Traceability

- Definitions & Processes
- Costs & Benefits
- Approaches

But what can Traces be used for?

- Impact Analysis
- Supporting Requirements Updates

Definition and Benefits

— — —

“Software change impact analysis is the activity of the software maintenance process that determines possible effects of proposed software changes”

- De Lucia, Fasano and Oliveto

Helps to

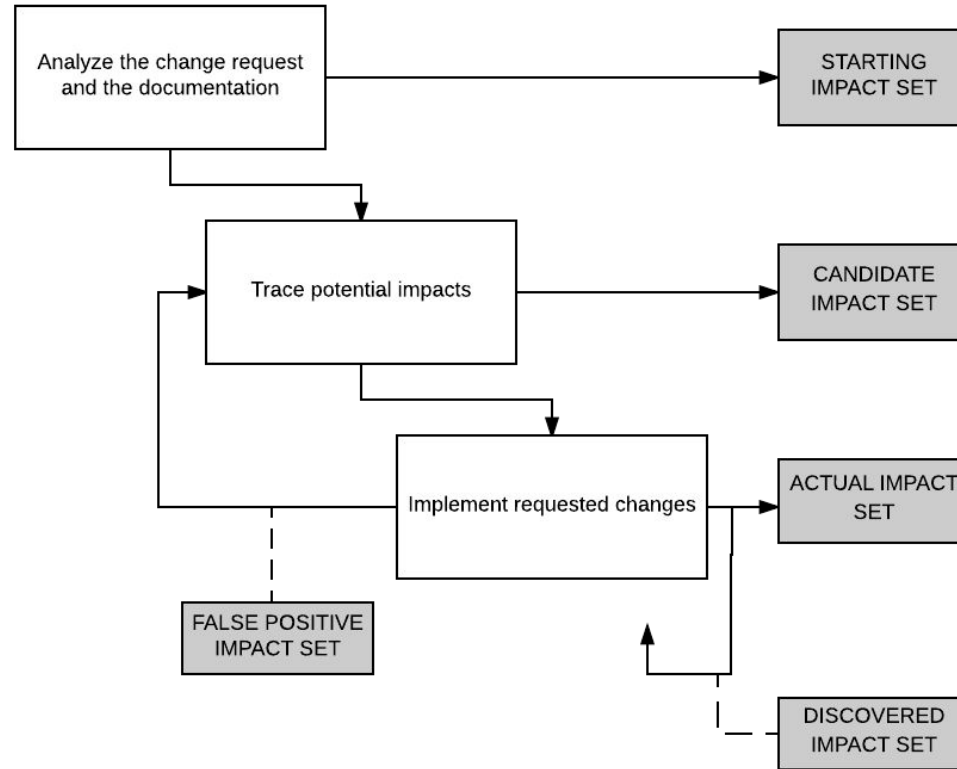
- Identify ripple effects
- Consider trade-offs between different approaches

Should consider

- Documentation abstracted from source code
- Prior changes, both successful and unsuccessful

Impact Analysis Process

— — —



The Role of Traceability

— — —

Goal of Impact Analysis Process is to identify a Candidate Impact Set which is very close to Actual Impact Set

→ Traceability allows for identifying affected Artifacts

→ Quality of Traceability therefore has direct influence on Impact Analysis

Issue

— — —

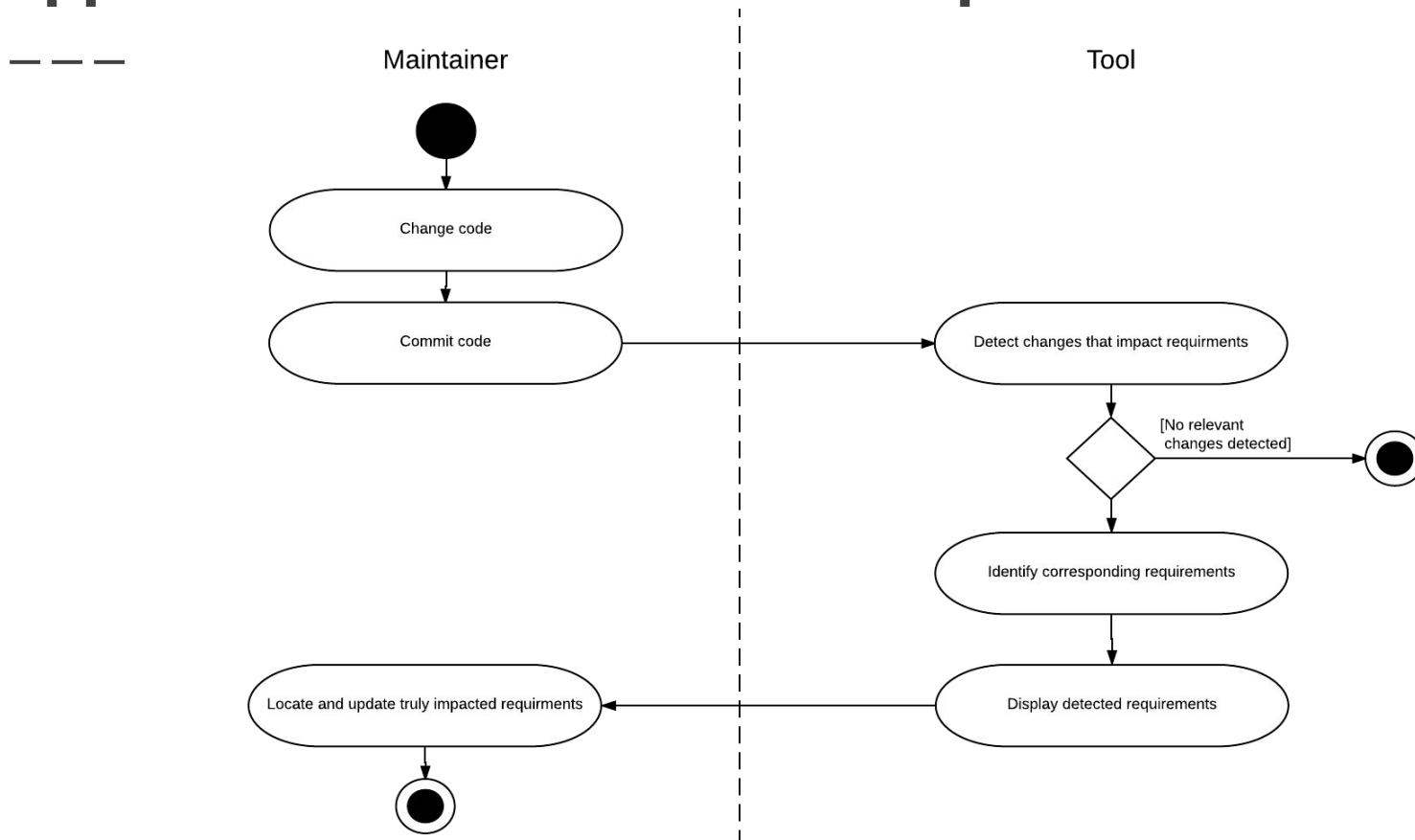
Requirement Specifications often include important information for maintenance and evolution tasks (e.g. rationale behind decision)

However, updating requirements is often a manual and expensive task

Therefore, requirements are seldom updated when software systems evolve

→ Requirement Specification becomes obsolete and useless!

Approach for Co-Evolution of Requirements



Approach for Co-Evolution of Requirements

— — —

Three Steps

1. Old and new Code is compared for changes that impact requirements
2. Terms describing changes are extracted from code and traced to the requirements
3. Impacted requirements are displayed to maintainer

→ Maintainer does not have to manually browse potential requirements

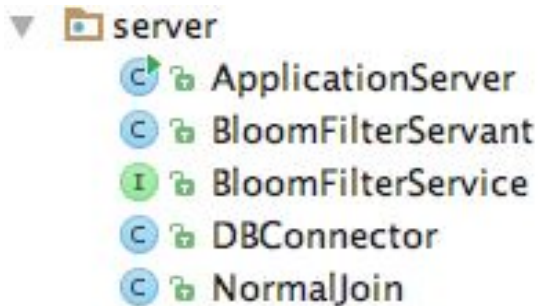
1. Heuristics to identify source code change

— — —

1: changes in method bodies $\Rightarrow$ refactoring/bug fixing; no impact

```
public void add_in_bloom(int value){  
    for(int i = 0; i < numberOfBloomFunctions; i++) {  
        int hashLocation = bloomFunction(value, i);  
        bitSet.set(hashLocation);  
    }  
}
```

2: addition of new elements(classes, methods, package) $\Rightarrow$ extension of features; impact



Identify source code change with heuristics

— — —

3: additions and removals of elements with similar names $\Rightarrow$ rename operations; no impact

```
public void add_in_bloom(int value){  
    ...  
}
```



```
public void addToBloom(int value){  
    ...  
}
```

4: changes in method signature (parameter, access rights) $\Rightarrow$ refactoring; no impact

```
public void addToBloom(int value){  
    ....  
}
```



```
private Bitset addToBloom(Bitset bitset, int value){  
    ...  
}
```

Identify source code change with heuristics

— — —

5: changes in private elements $\Rightarrow$ possible impact on system behavior

```
private int bloomFunction(int value, int functionNumber) {  
    return (baseFunction1.hash(value, slotCapacity) + (functionNumber+1) * baseFunction2.hash(value, slotCapacity)) % slotCapacity;  
}
```

6: additions of methods with same name, but different parameter (Overloading) $\Rightarrow$ same feature

```
public void addToBloom(int value){  
    ....  
}
```

```
private void addToBloom(Bitset bitset, int value){  
    ...  
}
```

2. Keyword extraction

— — —

Changed element	Names	Documentation	Call hierarchy
Package	Package, sub-classes	Package	No
Class	Class, sub-methods, sub-fields	Class	Yes
Method	Method, parent class	Method, parent class	Yes
Field	Field, parent class	Parent class	No

⇒ Element names & documentation contain the behavior and purpose.

⇒ Similarity computation with Vector space retrieval & tf-idf-weighting

Results

		AquaLush		iTrust		Connect	
Detecting relevant commits (Q1)	Accuracy	88.8		86.8		81.2	
	<i>TPR</i> (sensitivity)	100		83.8		100	
	<i>TNR</i> (specificity)	83.3		88.3		68.4	
Detecting outdated requirements (Q2)	<i>Cut rank</i>	<i>n</i> = 23	<i>n</i> = 59	<i>n</i> = 1	<i>n</i> = 7	<i>n</i> = 1	<i>n</i> = 3
	Recall	77.7	100	44.4	72	61.5	84.6
	Precision	21.7	9.6	44.4	9.7	61.5	28.2
	Fall-out	5.4	16	1.4	19	0.7	4.14

TPR, true positive rate; *TNR*, true negative rate.

⇒ high recall, but low precision

Conclusion

— — —

Traceability enables impact analysis

- Quality is influenced by quality of traceability

Automatic approaches enable co-evolution of requirements

- Not all changes affect requirements
- Drastically reduces time required to identify and update requirements

Discussion

— — —

- Do you think a approach achieving high recall is to prefer before a approach with high precision, but lower recall?

⇒ a tool returning not all affected requirements, but every returning is affected

Discussion

— — —

- Has anyone used traceability for impact analysis or co-evolution of requirements? If so, what was the experience like?
If not, how are the subjects handled?

Discussion

— — —

- Even though there are clear benefits in investing into requirement traceability, it still seems to be seldom used in practice. How can this be overcome?

References

— — —

- Charrada, E.B, Kozirolek, A. and Glinz, M. *Supporting requirements update during software evolution*. Journal of Software: Evolution and Process 27.3 (2015): 166–194
- De Lucia, A., Fasano F., and Oliveto, O. *Traceability Management for Impact Analysis*. Frontiers of Software Maintenance, 2008. FoSM 2008. IEEE, 2008.