

Assignment 5, Theme C: Impact Analysis with Traceability

University Zürich, Fall 2011
Requirements Engineering II

Group 3:

Andrei Andriesi [presenting]

Michael Blöchlinger

Kevin Caluser [presenting]

Andrei Cojocariu

Kristin Kruse

Outline

1. **Why** traceability?
2. **When** to use traceability?
3. **How** to use traceability?
 - Practices
 - Processes
4. **With what?**
5. **Conclusion**

1. Why traceability?

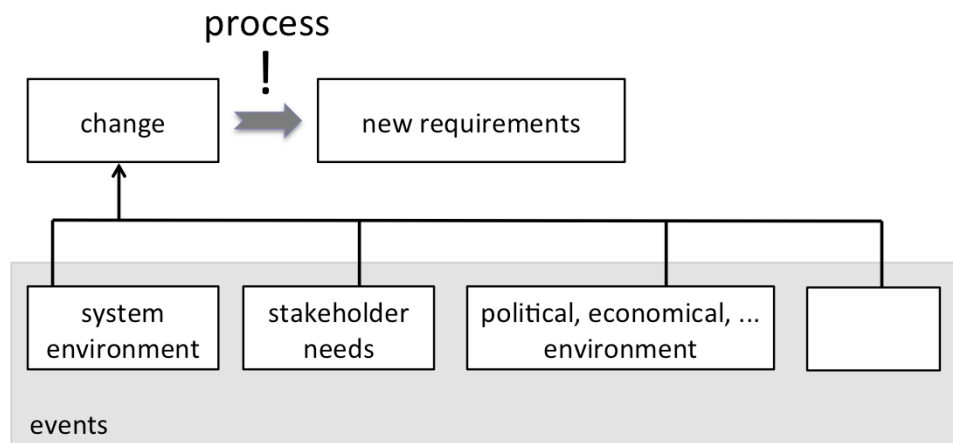
Laws of software evolution



A system must be continually adapted, or it will be progressively less satisfactory in its environment.

Assumption: Changes to software often initiated by changes to requirements

2. When traceability?

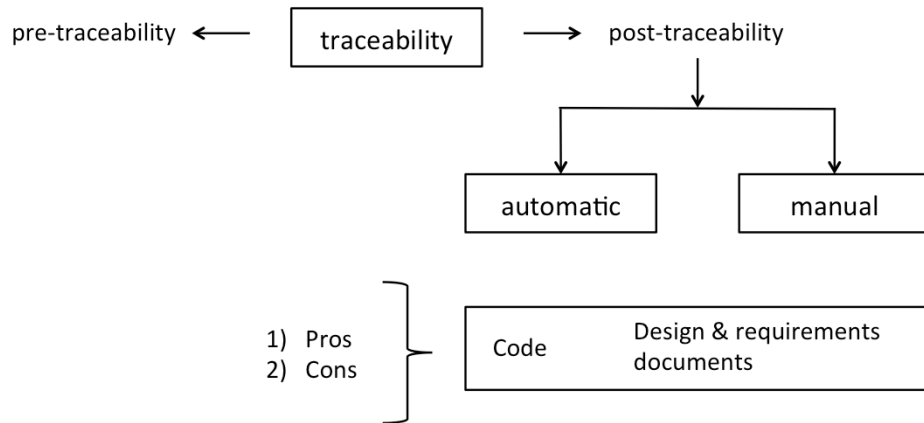


Factors that inflict changes to requirements during initial development or software evolution

*“Change is not a kick in the teeth,
unmanaged change is”*

Leszek Maciaszek

3. How to use traceability? - Practices



automatic:

algorithmic prediction of impacts. For example, relationship graphs for requirements [...] can be used with graph algorithms to identify the impact a proposed change would have on the system. The prerequisite for automatable strategies is a structured specification of the system.

manual:

no algorithm behind, but rather exploits experience of developers / Requirements Engineers

Post-traceability: link requirements to design and implementation

requirement → implementation (forward traceability)

implementation → requirement (backward traceability)

Code: is always up-to-date (whereas requirements documents are often outdated). Traceability within source code has always been around.

Requirements documents on the other hand provide a better overview for non-developers. Moreover, changes in software usually start with a change in requirements. So requirements documents should be considered when a change occurs.

3. How to use traceability? - Practices

- Traceability helps finding **dependencies** among requirements!
- Traceability helps you analyze the **impact of a change** in requirements.
 - How many components / requirements need to be changed?
 - Cost and labour estimation?
 - Is change feasible?

3. How to use traceability? - Processes



Plan for change involves recognizing the fact that changes occur, and that they are a necessary part of the system's development. This preparation is essential for changes to be received and handled effectively.

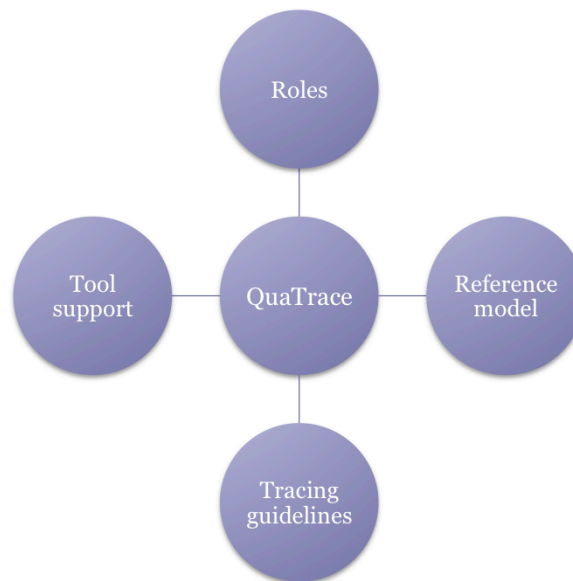
Baseline requirements means to create a snapshot of the current set of requirements. The point of this step is to allow subsequent changes in the requirements to be compared with a stable, known set of requirements.

A **single channel** is necessary to ensure that no change is implemented in the system before it has been scrutinized by a person, or several persons, who keep the system, the project and the budget in mind. In larger organizations, the single channel is often a change control board (CCB).

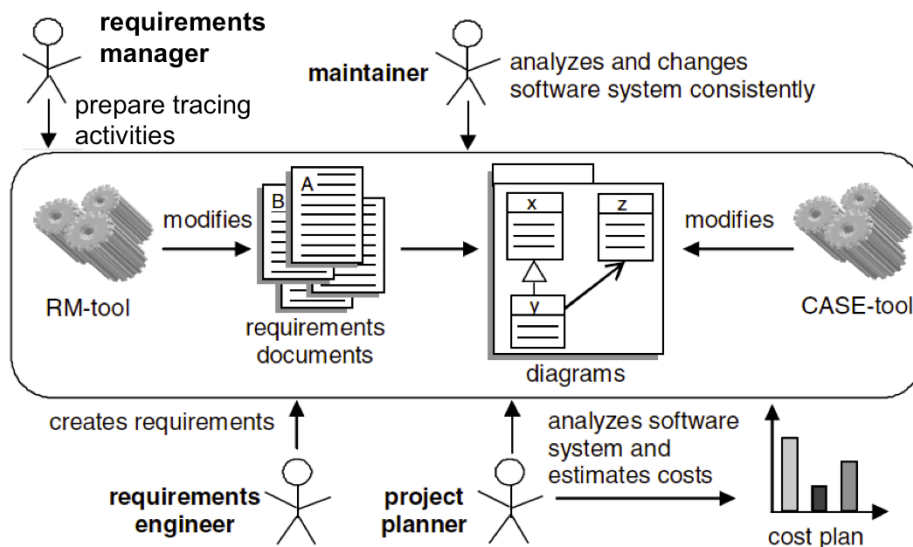
A **change control system** allows the CCB (or equivalent) to gather, track and assess the impact of changes. According to Leffingwell and Widrig, a change must be assessed in terms of impact on cost and functionality, impact on external stakeholders (for example, customers) and potential to destabilize the system. If the latter is overlooked, the system (as pointed out earlier) is likely to deteriorate.

To **manage hierarchically** defeats a perhaps too common line of action: a change is introduced in the code by an ambitious programmer, who forgets, or overlooks, the potential effect the change has on test cases, design, architecture, requirements and so on. Changes should be introduced top-down, starting with the requirements. If the requirements are decomposed and linked to other SLOs (Software Lifecycle Object, pretty much every object that occurs while developing software. E.g. requirements, classes, diagrams, etc.), it is possible to propagate the change in a controlled way.

3. / 4. Process example: QuaTrace - Overview



3. Process example: QuaTrace - Overview



VL Requirements Engineering II, Assignment 4 : Impact Analysis with Traceability

28.11.2011

Requirements Engineer

Make requirements & establish traces according to guidelines

Product Planner

Analyzes traces in order to estimate cost and effort

Maintainer

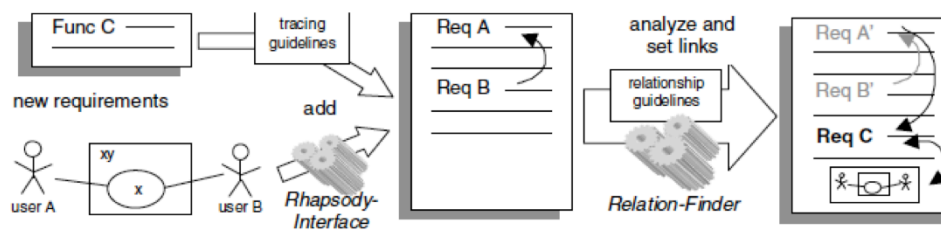
Integrate changes into the system

Requirements Manager

Prepare tracing activities within organisation (guidelines, reference models, etc.)

3. / 4. Process example: QuaTrace – New requirement

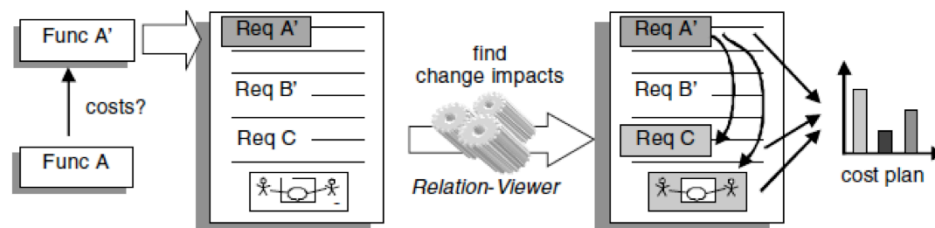
Perspective of the **Requirements Engineer**



The **requirements engineer** created the “*system requirements document*” with support of *RequisitePro* (*Requirements Management Tool*). Now, s/he documents the new functionality *Func C*, in form of a natural language paragraph and integrates a use case into a use case diagram with the help of *Rhapsody* (*CASE Tool*). Then, s/he exports the use *RequisitePro*. Then, s/he applies the relationship and tracing guidelines to establish traces between related requirements (e.g., a trace between the natural language paragraph and the use case that describe both the functionality *Func C* is established by using the same names). Then, s/he extracts traces (e.g., the name trace between the natural language paragraph and the use case) by using *Relation-Finder*.

3. / 4. Process example: QuaTrace – Changing requirement

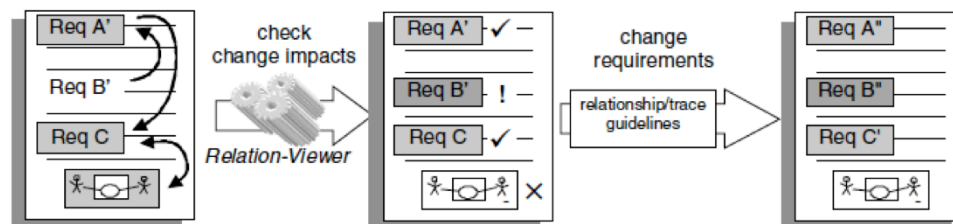
Perspective of the **Project Planner**



The functionality *Func A* has to be changed. The **project planner** analyzes the corresponding requirement *Req A'* in the requirements document. For performing an impact analysis, s/he uses the tool *Relation-Viewer*. It offers him a fine-grained impact analysis, helping him rating the effort and costs for changing *Func A* more precisely. Finally, s/he estimates the costs for the planned change.

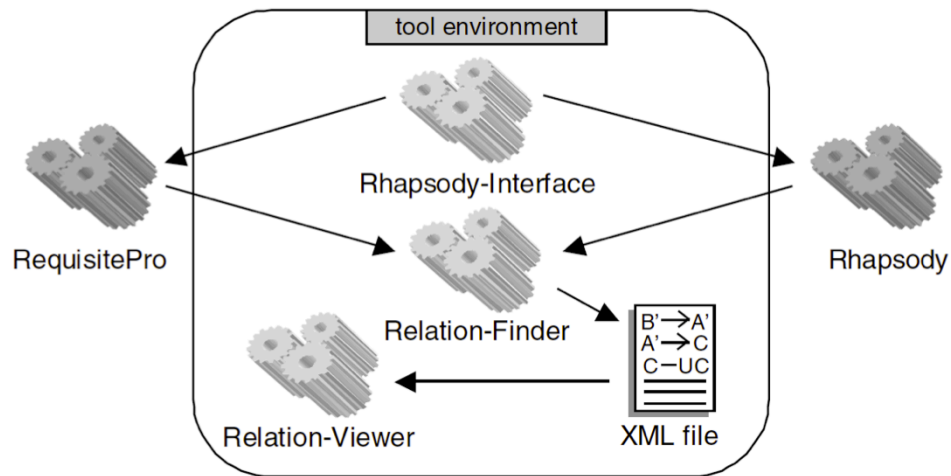
3. / 4. Process example: QuaTrace – Changing requirement

Perspective of the **Maintainer**



The **maintainer** integrates an accepted change to *Func A*. The impact analysis of the project planner resulted in the change impacts *Req A'*, *Req C* and the use case. The maintainer checks whether these impacts are really affected by the change with support of the *Relation-Viewer* and identifies that only *Req A'* and *Req C* have to be changed. When changing the change impacts with the help of *RequisitePro*, s/he analyzes and identifies a trace between *Req B* and *Req A* with the tool *Relation-Viewer*. Because of the modified interface of *Req A'*, *Req B* has to be changed, too.

4. With what?



Sample architecture of a Traceability tool support.

RequisitePro: Requirements Management Tool. All the requirements in plain text are stored in there)

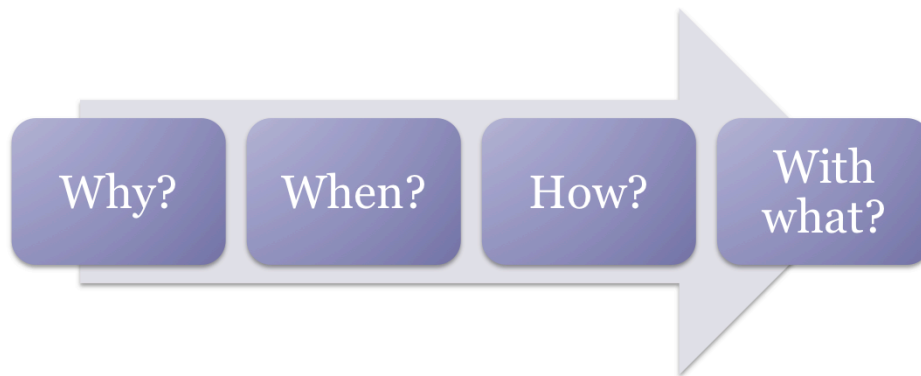
Rhapsody: Computer Aided Software Engineering Tool. Create and store (UML) diagrams and other architectural elements

Rhapsody Interface: helps linking Rhapsody objects into RequisitePro, where the tracing can be done.

Relation-Finder: Relation-Finder uses lexical analysis in order to find traces between Rhapsody objects and requirements from RequisitePro

Relation-Viewer helps to effectively display relations.

5. Conclusion



Interaction analysis through Requirements Traceability helps to

- estimate affected components / requirements upon change in requirements.
- estimate cost / effort of a change in requirements
- improves maintainability / understanding of software!

Requirements Traceability is a collaborative process

Establishing guidelines is crucial!!!!

- reference models
- tracing guidelines

Tool support helps a lot.