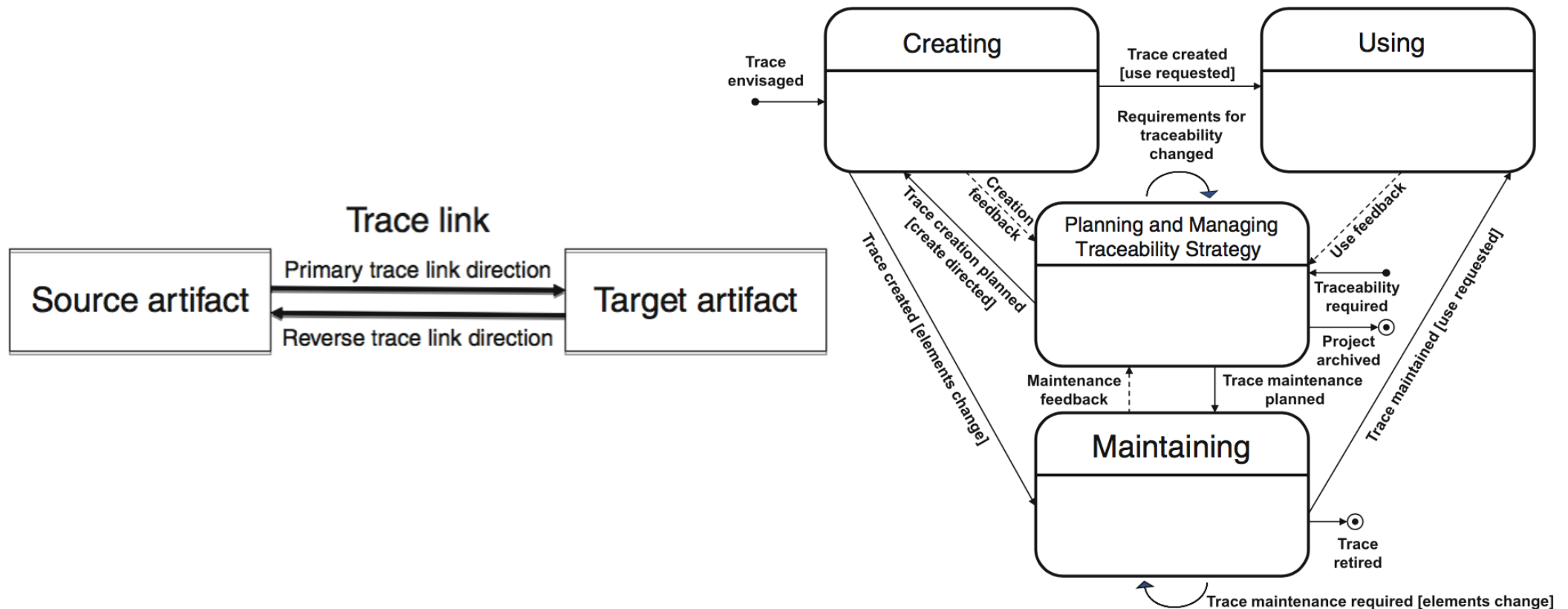


Automated traceability

Mario Viso & Florian Fischer

Traceability and Requirements Evolution



O. Gotel et al. (2012)

Traceability and Requirements Evolution

- Traceability of requirements is defined by the possibility to reconstruct through which phases and activities a requirement had to go through.
- A trace is a triple construct that involves a source artifact, a target artifact and a trace link, which is the association between the two artifacts. A trace artifact can be a single requirement, a group or cluster, a class or a person.
- Traces follow four main activities throughout their existence: creation, usage, planification & management of the traceability strategy and maintenance.
- The model depends on the traceability strategy, depending on the stakeholder's needs and continuous feedback.

O. Gotel et al. (2012)

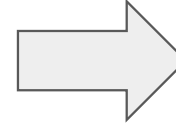
Automated traceability

What is requirement traceability for ?

- conduct impact analysis
- ensure sufficient test coverage
- certification and safety
- track project progress
- reconstruct decisions
- avoid rework
- ...



Needed for many activities in
system development !



Automated
traceability

Ingram & Riddle (2012)

Automated traceability

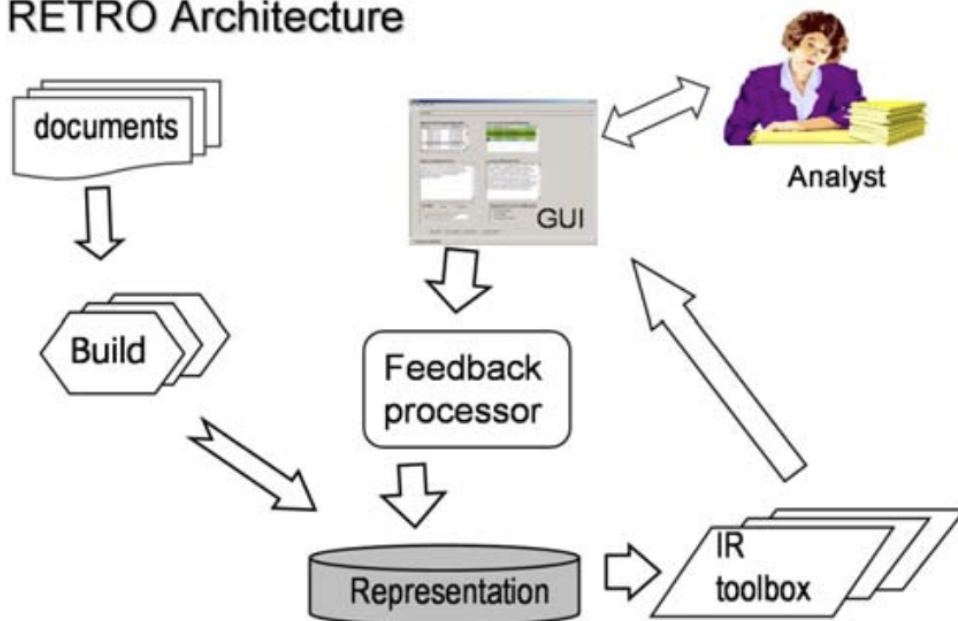
- Many activities encountered in any system development project require traceability to be conducted efficiently. However, ensuring traceability comes at high costs and is often ignored when budget restrictions apply.
- There is a trade-off between the cost of traceability and the extra costs implied by a lack of traceability.
- A potential way to make this trade-off easier to solve is to reduce the costs of creating and maintaining traces.
- Automation techniques, many from the field of information retrieval, can help generating automated traces.

Ingram & Riddle (2012)

Improving software maintenance through traceability recovery

(Hayes et al., 2007)

RETRO Architecture



ID	Feature
1	Tracing entire dataset at once
2	Tracing elements one at a time
3	Filtering toolbar
4	Filtering option to show top number of links
5	Killing the links that are hidden by the filter
6	Global filtering of candidate link lists
7	Local filtering of candidate link lists
8	View of low-level elements one at a time
9	View of low-level elements in document order
10	View of low-level elements in order of similarity
11	“Freezing” of high-level element tracing
12	Assignment of “Link” and “Not a link” to links
13	Feedback loop
14	Browse tab (Manual tracing mode)
15	Text search in browse tab
16	Adding links to the RTM from the browse tab

Improving software maintenance through traceability recovery

(Hayes et al., 2007)



	Recall	Precision	Total time (min)
Manual group (Group 1)	0.33	0.24	120.67
RETRO group (Group 2)	0.70	0.13	41.88
<i>T</i> test (<i>p</i> value)	0.001	0.01	0.0004

Improving software maintenance through traceability recovery

(Hayes et al., 2007)

- The authors present a tracing tool, RETRO, that automates the updating of requirements traceability matrix (RTM).
- RETRO consists of a set of information retrieval and text mining methods and an interface that was iteratively improved together with users.
- In a validation case-study, 30 students were compared in their tracing abilities. They were divided in two groups, one doing the task manually, the other one with RETRO.
- The results show that the RETRO group was 3 times faster and had a twice higher recall rate (traces retrieved as relevant / total relevant traces) for a slightly lower precision rate (traces retrieved as relevant / total retrieved traces)

Best Practices for Automated Traceability

(Cleland-Huang et al., 2007)

- Automated traceability relies on VSM- and PN-based algorithms to compute the likelihood of a link by removing stopwords (“this”, “the”,..) from the artifact text and stemming remaining to their root forms. In addition, in code & UML docs it separates method names (e.g. “getLastName()” -> “get last name”) and removes program-related keywords such as “while” and “for”.
- Rarer index terms are more relevant than common ones in detecting potential links.
- Evaluation of experimental results using two metrics:
 - **Recall** measures the extent to which all desired links are retrieved.
 - **Precision** measures the percentage of received relevant links.
- In automated traceability returning a good set of candidate traces requires properly structuring and specifying the artifacts.

9 Best Practices (Cleland-Huang et al., 2007)

Traceability Environment

1. Trace for a purpose
2. Define a suitable trace granularity
3. Support in-place traceability

Creating traceable artifacts

4. Use a well-defined project glossary
5. Write quality requirements
6. Construct a meaningful hierarchy
7. Bridge the intradomain semantic gap
8. Create rich content

Introducing automated trace practices

9. Use a process improvement plan

Best Practices for Automated Traceability

(Cleland-Huang et al., 2007)

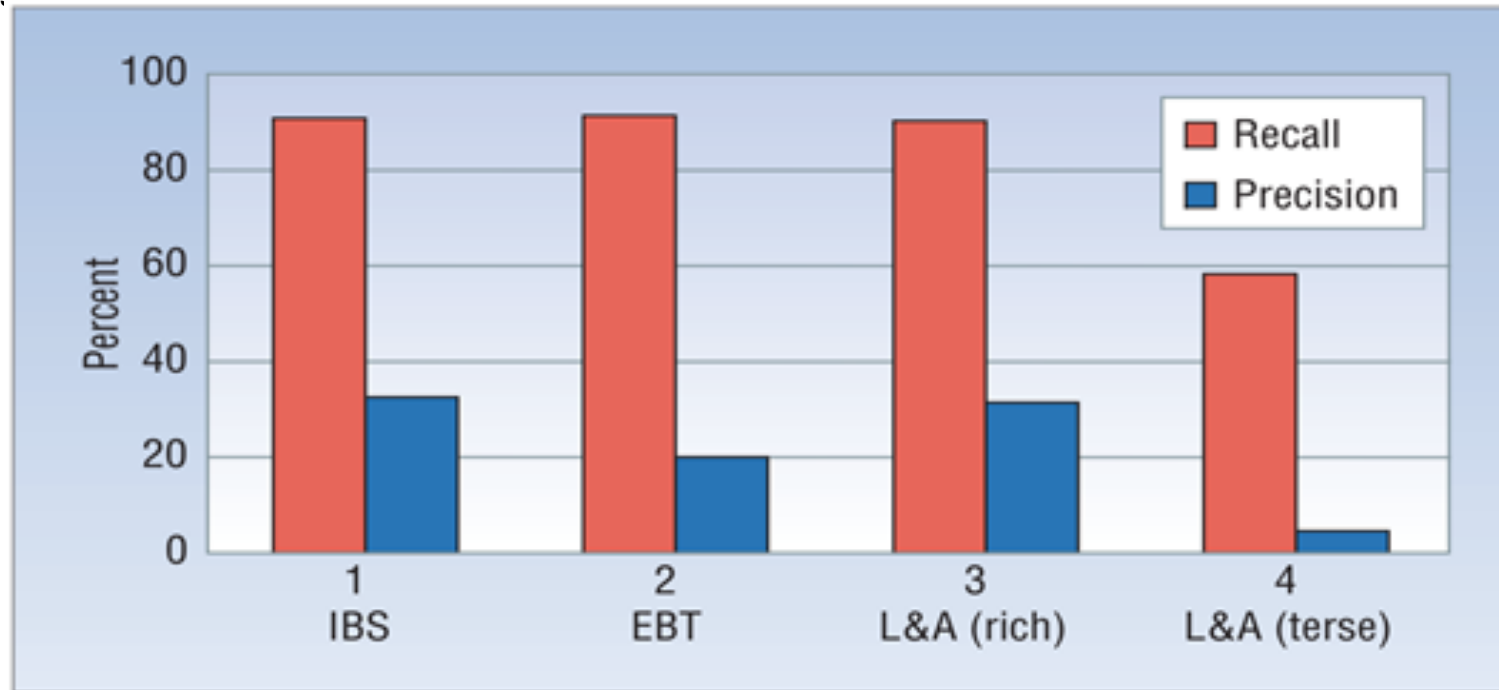


Figure 3. Trace retrieval results for several data sets. Data sets 1-3 all returned acceptable precision results ranging from 19 to 32 percent when recall was fixed close to 90 percent, while data set 4 returned an abysmal precision rate of 4 percent at recall levels of 58 percent.

9 Best Practices (I/II)

Traceability Environment

1. Trace for a purpose: Making up-front decisions and explicitly modelling the traceability needs of the project stakeholders provide the necessary physical infrastructure to support automated traceability.
2. Define a suitable trace granularity: Effort needed to create links vs Value returned through tracing: A x10 increase in granularity returned only a x2 improvement in precision. To tackle this automated traceability tools can generate traces to meaningful related grouped sentences.
3. Support in-place traceability: Developers generally use many third-party case tools to manage artifacts, and traceability should be provided to these artifacts as they reside within their native environments. This is referred as in-place traceability. An in-place traceability tool that provides advanced filters is essential.

9 Best Practices (II/II)

Creating traceable artifacts

4. Use a well-defined project glossary: A well-constructed project glossary, initially defined and used consistently throughout product development, will increase consistency in term usage and improve traceability and the quality of retrieved traces.
5. Write quality requirements: The qualities of completeness and conciseness are important for automated traceability. A best practice for requirements in general, but requires requirements specification training.
6. Construct a meaningful hierarchy: Headings or meaningful package names, can enable a trace retrieval tool help to construct more accurate links.
7. Bridge the intradomain semantic gap: Defining intradomain synonyms and using a tool that supports domain-specific synonym matching
8. Create rich content : Enhancing terse or hard-to-trace requirements with supporting rationale and domain information.

Introducing automated trace practices

9. Use a process improvement plan: Studies suggest that 20-35 percent precision should be achievable at recall levels of 90 percent. Where necessary, training and process improvement initiative plan improve results.

Questions for discussion

- Can you think of any other part of Requirements Engineering which can be automated?
- What risks in terms of Automated Traceability can you imagine? Or are there only benefits?
- In what cases could Automated Traceability be more expensive than Manual Traceability?

References

J. Cleland-Huang, Settimi, R., Romanova, E., Berenbach, B., Clark, S., (2007). Best Practices for Automated Traceability. *IEEE Computer* 40(6): 27-35.

O. Gotel, Cleland-Huang, J., Hayes, J. H., Zisman, A., Egyed, A., Grünbacher, P., Dekhtyar, A., Antoniol, G., Maletic, J. and Mäder, P. (2012). Traceability Fundamentals. In J. Cleland-Huang, O. Gotel, A. Zisman (eds.): *Software and Systems Traceability*. Springer, London. 3-22.

J. Hayes, H., A. Dekhtyar, S. Sundaram, E. Holbrook, S. Vadlamudi, A. April (2007). REquirements TRacing On target (RETRO): Improving software maintenance through traceability recovery. *Innovations in Systems and Software Engineering* 3(3):193-202.

C. Ingram and S. Riddle (2012). Cost-Benefits of Traceability. In J. Cleland-Huang, O. Gotel, A. Zisman (eds.): *Software and Systems Traceability*. Springer, London. 23-42.