Managing the co-evolution of software artifacts

Managing the co-evolution of software artifacts

J.M.A.M. Gabriels[a], D.H.R. Holten[b], M.D. Klabbers[a],
W.J.P. van Ravensteijn[b], A. Serebrenik[c]

[a]Laboratory for Quality Software, Eindhoven University of Technology
[b]SynerScope B.V.
[c]Model-Driven Software Engineering, Eindhoven University of Technology

Software development projects are virtually always carried out under pressure. Planning and budgets are tight, room for errors is non-existent and the pressure to deliver is high. Natural questions for (test) managers arise, such as: "When have we tested enough?" and "How many tests do we have to redo for this new version?". The naive answer would be: "when we have convinced ourselves through testing that all requirements are satisfied". Unfortunately, attaining maximal confidence with minimal effort is not easy.

In order to convince ourselves that the system does what it is supposed to do, tests are needed. Requirements, design and code change during the development of software. As a consequence, tests need to change as well. In the end we want to ensure that all requirements and risks are adequately addressed with tests. For this, tests at different levels of abstraction and for different software artifacts are required and need to be managed.

Traceability matrices are often used to relate user requirements, design, code, and tests. Traceability allows to link elements from different software artifacts, like requirements, design components and code components, to each other and to test cases. As a result, traceability can be used to analyze for example how well software artifacts are covered by test cases. Because a requirement leads to design components and eventually to code, tests are needed at each stage. Traceability can tell us how well test cases cover different software artifact elements. This information can be used to uncover mistakes in software artifacts at an early stage and actively manage the development and test efforts. Unfortunately, traceability information is often spread out over multiple artifacts and describes only the current situation.

TraceVis, a visual analytics tool based on the master thesis of Van Ravensteijn[1] combines the traceability information of multiple software artifacts in an interactive way. The tool was recently applied to fraud detection in financial transactions [2] and software model transformations[3], Figure 1 shows the traceability between four, vertically placed, hierarchical software artifacts: acceptance test plan (ATP), user requirements document (URD), software requirements document (SRD), and architectural design document (ADD). Hierarchy within each document is given by its division into chapters, sections and subsections. Each line between hierarchies represents a link between elements of two artifacts, e.g., user requirement being tested by an acceptance test or architectural component implementing a software requirement. The hierarchies can be collapsed and extended and risk levels and priorities can be

[1] W.J.P. van Ravensteijn, *Visual traceability across dynamic ordered hierarchies*, M.Sc. thesis, 2011, Eindhoven University of Technology
[2] SynerScope on-line demos: http://www.synerscope.com/demos, and, specifically, *SynerScope for Fraud* http://www.synerscope.com/content/SynerScope%20for%20Fraud1.pdf
[3] M.F. van Amstel, A. Serebrenik, & M.G.J. van den Brand, *Visualizing traceability in model transformation compositions*, 2011, Workshop on Composition and Evolution of Model Transformations, London: Department of Informatics, King's College London.

visualized by giving the elements different colors. The edge bundling technique bundles similar relations in the middle, clearly showing deviations. Furthermore, TraceVis provides a way of assessing the evolution of traceability between artifacts though a timeline (lower part of Figure 1). The timeline shows such events as addition, modification or removal of individual elements, e.g., user requirements or tests, as well as addition or removal of traceability links between the elements.

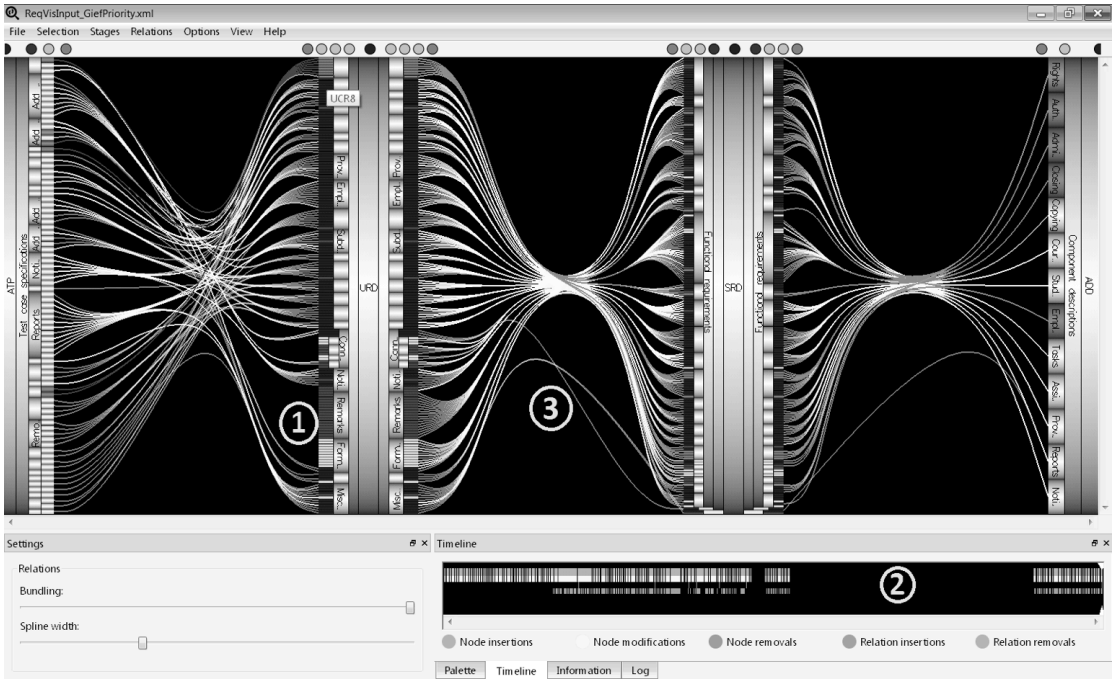

Figure 1: *TraceVis tool with traceability information from a student capstone project at the Eindhoven University of Technology. The edge bundeling technique makes it easy to spot deviations.*

Already at first glance, we can see points of attention in Figure 1: a gap in requirement coverage (1) and a gap in the timeline (2). The gap labeled (1) shows some medium and low priority user requirements not covered by acceptance tests. The gap labeled (2), located in the timeline, shows that test cases were added very late in the project. The gap itself relates to implementation activities in which there were no changes to the shown software artifacts.

By interactively inspecting the traceability information, we can discover points of interest. Figure 1 reveals, for example, that a selected user requirement (URC8) is tested by one acceptance test, corresponds to one software requirement and is implemented in one architectural component. While this does not seem to be problematic, further inspection of the evolution of user requirements closely related to URC8 tells an entirely different story. While grouped together in the URD, the corresponding software requirements are spread all over the SRD, and implementation involves five out of thirteen architectural components.

Figure 1 also shows outliers; grey lines running off-center between the URD and the SRD (3), and between the SRD and the ADD. The core part of the SRD consists of two parts: Requirements and Rights table. The Requirements part consists of Functional requirements further divided in groups (with 189 requirements in total) and Non-functional requirements (1 requirement). The Rights table part contains only one element, i.e., the rights table. This means that individual functional requirements are nested at depth four, the non-functional requirements at depth three and the rights table at depth two. Therefore the rights table is an “outlier” in the organization of the SRD.

The evolutionary traceability information allows us to see how well tests cover artifacts and whether risks are sufficiently tackled. It gives insight in the balance between tests, priorities, and risks and can support decision making in assigning test effort. Furthermore, it can help in determining which tests need to be redone when a certain component or requirement changes.
The insight in the co-evolution of software artifacts and associated tests makes it possible to actively manage test effort from an early stage on.

About the authors

Joost Gabriels (j.m.a.m.gabriels@tue.nl) received his M.Sc. in Computer Science from the Eindhoven University of Technology (TU/e) in 2007. Currently, he is a researcher and consultant at the Laboratory for Quality Software (LaQuSo), TU/e. His interests include software architecture and design, specification languages, program verification and the quality of software in general.

In his Ph.D., defended in 2009 at TU/e Dr. **Danny Holten** (danny.holten@synerscope.com) has developed techniques for the visualization of graphs and trees. He received the best paper award at IEEE INFOVIS 2006 and was nominated for the best paper award at ACMCHI 2009. He furthermore received the TU/e Doctoral Project Award for the best PhD dissertation in 2009. From 2009 to 2011, he worked as a postdoctoral researcher. Since April 2011, Danny Holten is CTO at SynerScope B.V., a Dutch visualization-research-inspired TU/e spin-off company that leverages his PhD research for "Big Data" analysis.

Martijn Klabbers (m.d.klabbers@tue.nl) received a M.Sc. in Computer Science from the Technical University Delft, The Netherlands in 1995. At the Technical University of Eindhoven, he is a senior consultant at LaQuSo (Laboratory for Software Quality). His research interests include certification, decision support systems, and user requirements.

In 2011, **Wiljan van Ravensteijn** (wiljan.van.ravensteijn@synerscope.com) received his Master's degree in computer science (with honors) from the Dept. of Mathematics & Computer Science at TU/e. His Master's thesis was titled "Visual traceability across Dynamic Ordered Hierarchies". Since August 2011, Wiljan van Ravensteijn is Software Developer at SynerScope B.V.

Dr. **Alexander Serebrenik** (a.serebrenik@tue.nl) is an assistant professor of Model-Driven Software Engineering at TU/e. He has obtained his Ph.D. in Computer Science from Katholieke Universiteit Leuven, Belgium (2003) and M.Sc. in Computer Science from the Hebrew University, Jerusalem, Israel. Dr. Serebrenik's areas of expertise include software evolution, maintainability and reverse engineering, program analysis and transformation, process modeling and verification.