Managing the co-evolution of software artifacts

ir. J.M.A.M. Gabriels
dr. ir. D.H.R. Holten
ir. M.D. Klabbers
ir. W.J.P. van Ravensteijn
dr. A. Serebrenik

synerscope
connecting the dots

LaQuSo
Laboratory for Quality Software

TU/e
Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

LaQuSo, SynerScope

- **Laboratory for Quality Software**
  - Part of TU/e department of Mathematics and Computer Science
  - Valorization of academic knowledge for use in business and industry
  - Feeding academia with questions from industry

- **SynerScope**
  - Spin-off from TU/e's Visualization group with the creators of TraceVis
  - Invented the edge bundling techniques
  - Discovery of patterns in Big Data
  - Successful application in financial transactions for detecting fraud

Difficult questions

- **Software development is carried out under pressure**
  - "Time to Market" vs. "Costs" vs. "Quality".
- **Questions arise such as:**
  - Have we tested enough?
  - How many test do we have to redo for the new version?
- **These are difficult questions to answer, but:**
  - Every requirement should be tested with an acceptance test
  - Ideally, requirements are as thoroughly tested as their perceived risk

How to go about questions like this?

Traceability

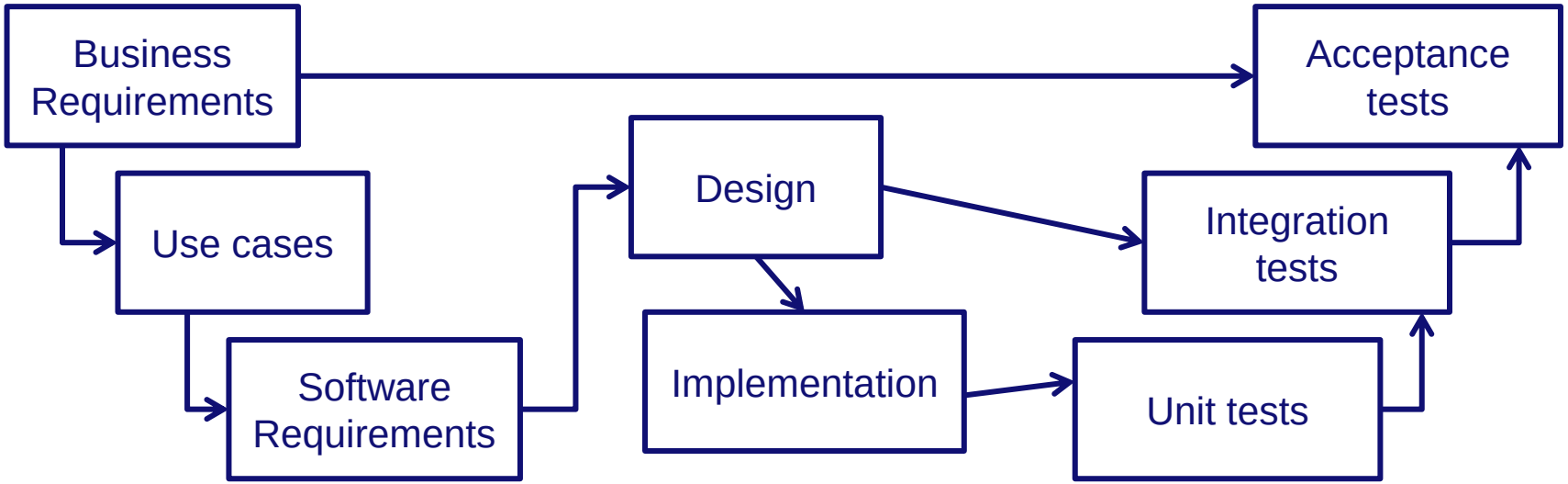
- **The usual answer is "*traceability*"**

	Req1	Req2	Req3	Req4	Req5	Req6	Req7	Req8
Test1	X							X
Test2	X							
Test3		X	X					
Test4				X				
Test5				X	X			
Test6					X			
Test7						X	X	

- **The "*traceability matrix*" *above* shows the traceability between requirements and test cases**

Traceability

- **Unfortunately, if we want full traceability across multiple software artifacts, things are more complex**

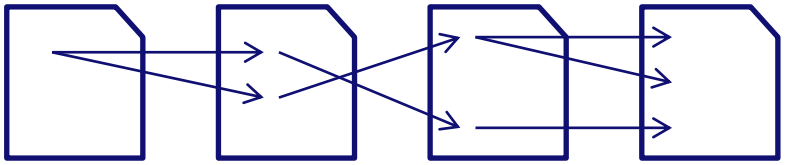


Traceability

- **There are different kinds of traceability relations:**
  1. **Direct relations between software artifacts**
     e.g. components (from design) vs. code units
  2. **Direct relations between units of a software artifact**
     e.g. business requirements vs. software requirements
  3. **Indirect (transitive) relations between software artifacts**
     e.g. business requirements vs. acceptance tests
- **To investigate the last category, we need to look at "*traceability chains*"**

Traceability

- **To investigate indirect relations using traceability matrices, we usually end up with:**
  - Multiple artifact elements in a big matrix (1-to-n relations)
  - Matrices spread over multiple artifacts



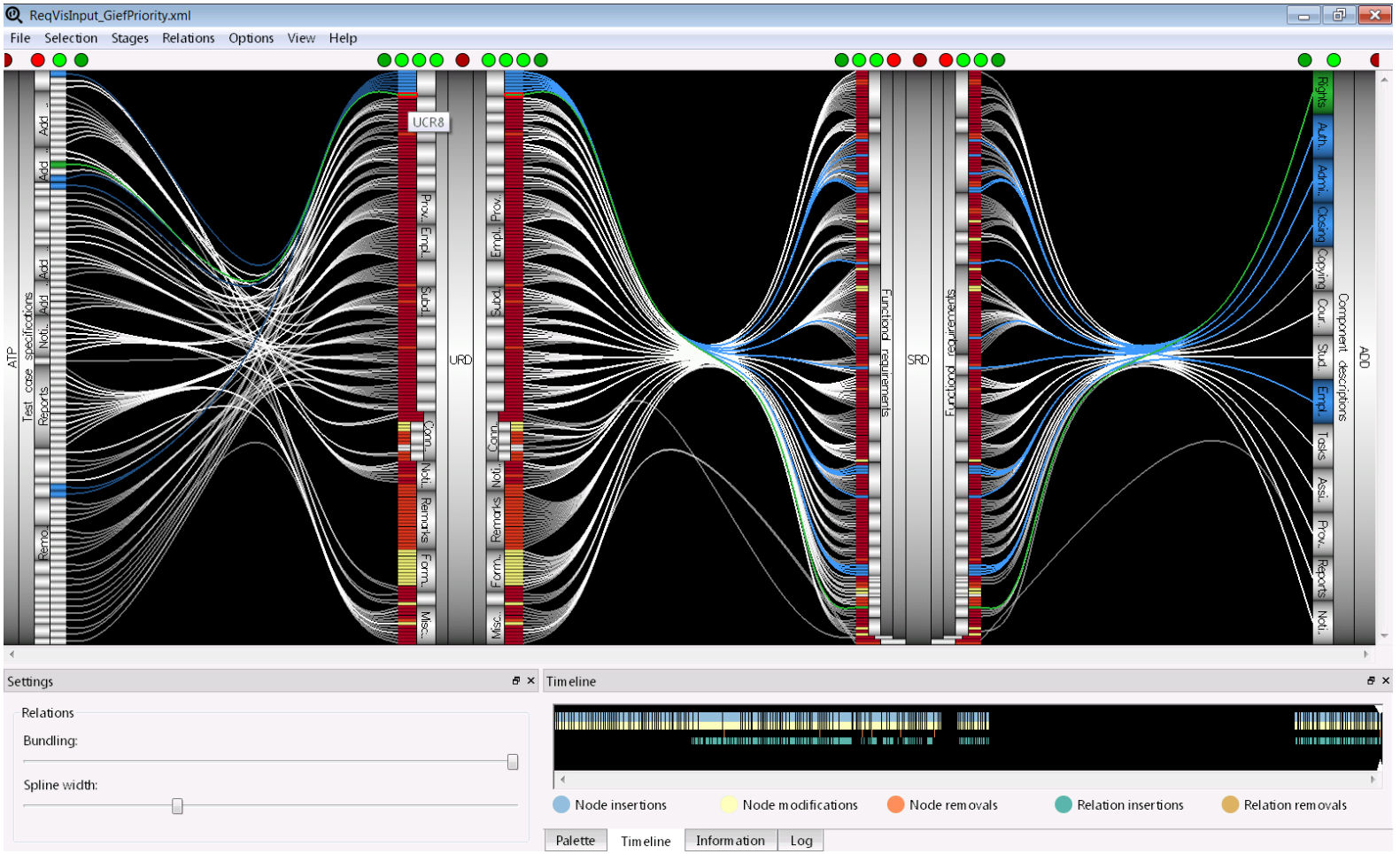
- **“Seeing things” and changing things in either of these setups can be challenging**
- **The effect of a change is difficult to assess**

Traceability with TraceVis

- **In 2011, Wiljan van Ravensteijn started TraceVis as part of his Masters Project at the TU/e**

- **The goal was to create a visual analytics tool to:**
  - Visualize traceability relations with hierarchies
  - Allow interactive manipulation of the view
  - Show the co-evolution of traceability across artifacts

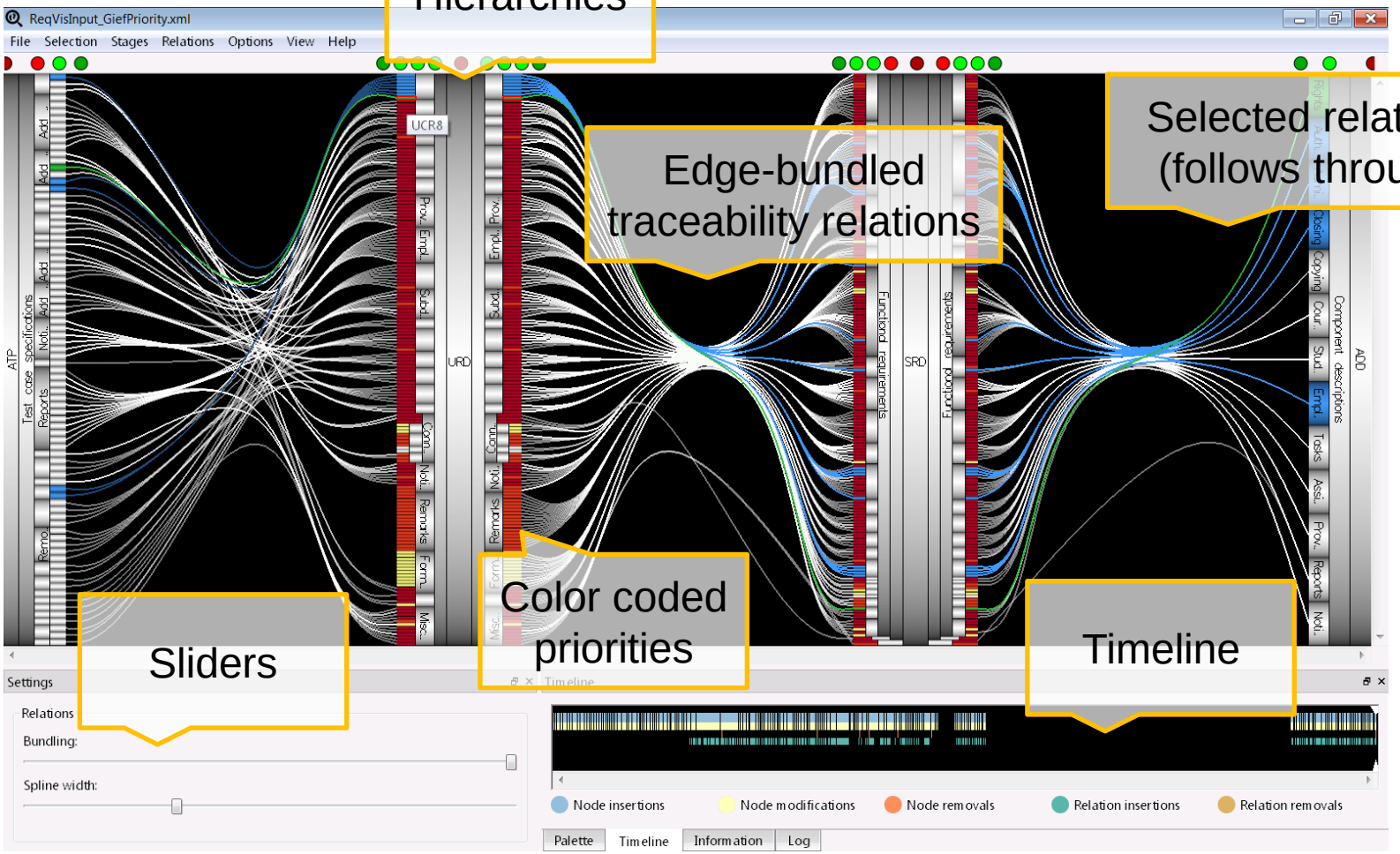
TraceVis



LaQuSo Laboratory for Quality Software

TU/e Technische Universiteit Eindhoven University of Technology

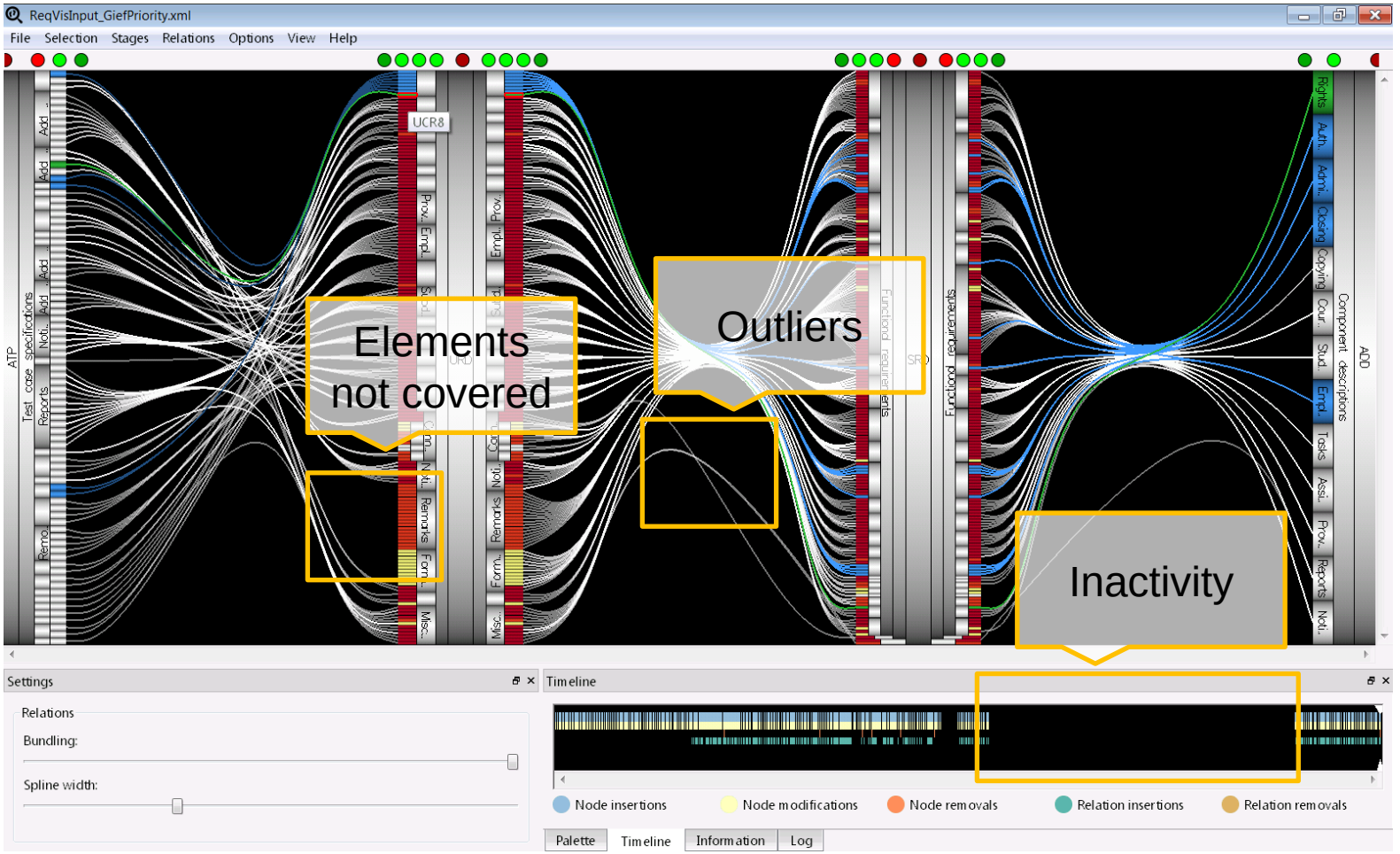
TraceVis - Overview



LaQuSo Laboratory for Quality Software

TU/e Technische Universiteit Eindhoven University of Technology

TraceVis - Patterns



LaQuSo Laboratory for Quality Software

TU/e Technische Universiteit Eindhoven University of Technology

Short Demo

Conclusions

- **With TraceVis, we can:**
  - Interactively visualize traceability relations at different levels
  - Spot anomalies such as outliers, empty spaces, inactivity, etc. without knowing in advance what they look like
  - Browse through the timeline and see how things evolve
- **We can get insight into:**
  - Coverage (or the lack thereof) between artifacts
  - Distribution of traceability relations between artifacts
  - The co-evolution of artifacts
  - The completeness of traceability chains

Future research

- **Often, the traceability information is (partly) missing:**
  - Investigate to what extent we can reconstruct traceability information from artifacts
    - Extract an *architecture proposal* from code
    - Extract unit test vs. code traceability from code
- **We want to optimally steer development / test effort:**
  - Investigate problems with traceability in practice
    - Visualizing industrial datasets
  - Extend TraceVis with additional analysis features

Thank you!

Questions / Feedback / Ideas?