Looking for Stability

LaQuSo
Laboratory for Quality Software

Joint Session Developer Track & Workshop on Software Quality, Net.Objectdays 2005
Erfurt (Germany)
September 22nd, 2005

Dr. Cornelis Huizing,
Dr. Ruurd Kuiper,
Dr. Ir. Teade Punter,
Dr. Alexander Serebrenik

LaQuSo is an activity of Technische Universiteit Eindhoven

LaQuSo

Objective

- Introduction
- Package decomposition
- Coupling
- Dependency Structure
- Code duplication
- Malpractices
- Conclusions

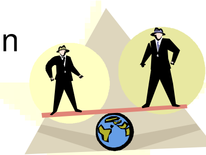
- Show how to evaluate (assess) software product quality
  - LaQuSo – Laboratory for Quality Software
    - Faculty of Computer Science and Mathematics, Eindhoven University of Technology
    - Verification and Validation of Software
- Examine the ability of existing tools (static analysis) to determine a particular software characteristic

LaQuSo

Stability

- Introduction
- Package decomposition
- Coupling
- Dependency Structure
- Code duplication
- Malpractices
- Conclusions

- ***Stability*** = capability of the software product to avoid unexpected effects from modifications of the software (ISO 9126)
- How to *assess* stability?
  - ISO-metrics require knowledge on
    - History of the modifications, and
    - Impacts of the modification
  - May be unavailable in practice
    - Discover instability *before* it ruins the software
  - Alternative operationalisation is required!



LaQuSo

Specification - 5 issues

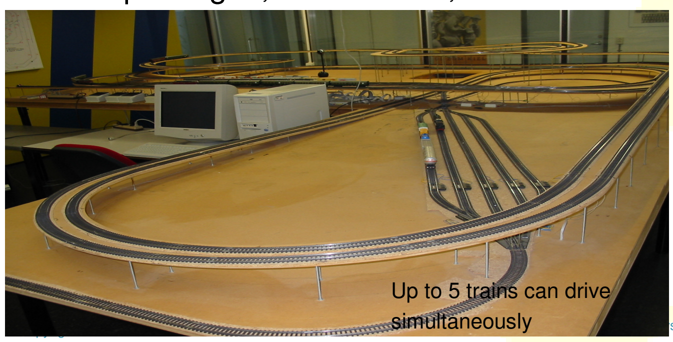
- Introduction
- Package decomposition
- Coupling
- Dependency Structure
- Code duplication
- Malpractices
- Conclusions

- <u>Our contribution:</u> stability-related issues
  - Design:
    - Functional decomposition
    - Coupling
    - Dependency structure
  - Implementation
    - Code duplication
    - Implementation malpractices
- *Assess stability by assessing these issues*
- Apply our approach to a case study.

LaQuSo

Case study

- Märklin toy railroad system
  - Developed by TU/e students
  - 8 students
  - Scheduling/security
  - 9 packages, 164 classes, 17828 lines of code



LaQuSo

Functional decomposition

- Division in a number of units
- Documentation vs. Implementation
  - Later changes based on the documentation can have unexpected effects!

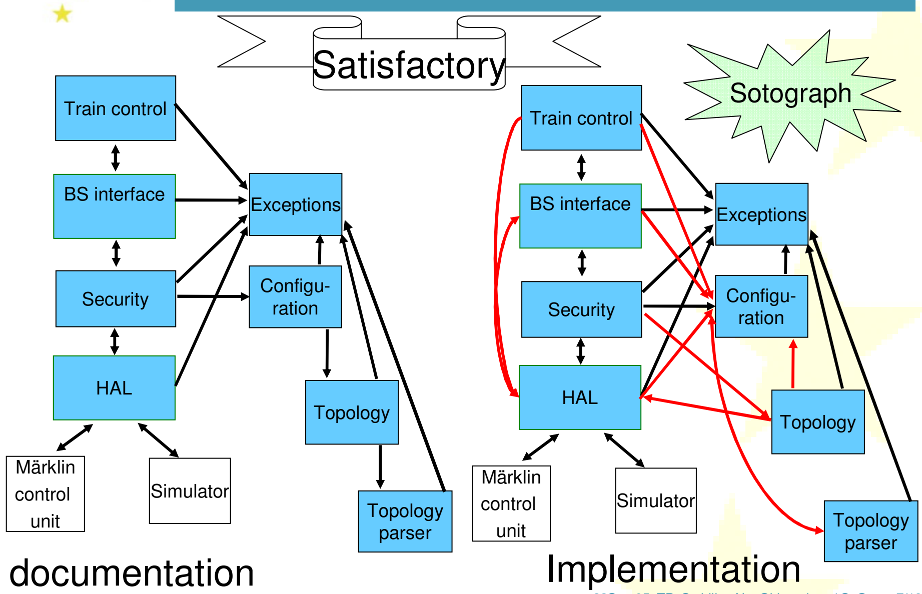
- Case study:
  - The same units are present.

  ✓ Good

Coupling

- Degree of interdependence between a pair of units
  - “Call” relations
- Documentation vs. Implementation
- Example tool: Sotograph
  - Visualization of internal structure of a system

Coupling: Case study

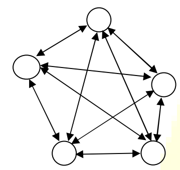


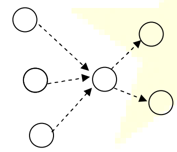
LaQuSo

Dependency structure

- Introduction
- Package decomposition
- Coupling
- Dependency Structure
- Code duplication
- Malpractices
- Conclusions

- Entire system of relations between packages and classes
- Architectural anti-patterns
  - Tangles
  - Global/local butterflies
  - Global/local breakables
  - Global/local hubs
- Propagation of change

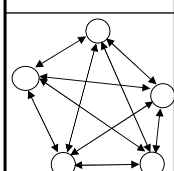
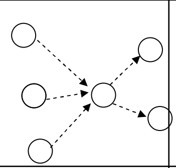
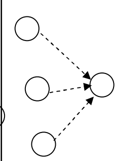
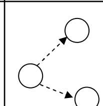




LaQuSo

Case study

- Introduction
- Package decomposition
- Coupling
- Dependency Structure
- Code duplication
- Malpractices
- Conclusions

Tangle	Global Hub	Global Breakable	Global Butterfly
			
4 tangles (longest – 24 elements)	30 (22%)	62 (45%)	90 (66%)

SA4J

Propagation of change

- Introduction
- Package decomposition
- Coupling
- Dependency Structure
- Code duplication
- Malpractices
- Conclusions

- Changes in one class can lead to changes in another class.
- Case study:
  - On average, when an element (class or package) is modified 46.3 other elements are affected (35%).
  - For stable programs this value < 10%.

Poor

Code duplication

- Introduction
- Package decomposition
- Coupling
- Dependency Structure
- Code duplication
- Malpractices
- Conclusions

- Presence of identical or almost identical code fragments
  - “Almost identical” – minor syntactical differences
  - Modification of a duplicated code should propagate to other clones
  - Some anti-patterns can be eliminated by duplication without improving the design
- Tools
  - IntelliJ IDEA 4.5
  - Gemini

Case study: Code duplication

- 27 duplication groups
  - Up to 18 lines of code
- Benchmark: InfoGlue
  - 153 clone groups
- CloneGroups(InfoGlue) : CloneGroups(Trains) ≈ Methods(InfoGlue) : Methods(Trains)

IntelliJ IDEA

Case study: Code duplication

- 70% of a file = clone of the remaining files
- Duplicated LOC = 1270, 7%.
- Kapser, Godfrey: on average: 5-10%.

Gemini

Satisfactory

LaQuSo

- Introduction
- Package decomposition
- Coupling
- Dependency Structure
- Code duplication
- Malpractices
- Conclusions

Implementation Malpractices (1)

- Programming practices that do not lead to an erroneous execution but can cause it when the program is modified.

ESC/Java

```
public boolean equals(Object switch) {
    return (getID() == ((Switch)switch).getID());
}
```

- Always called with `switch` instance of `Switch`
- Produces a casting error if called otherwise!
- `equals` was implemented 13 times
  - 10 times like above
  - 2 times `equals` always returns `false`
  - Implemented correctly only <u>once</u>!

LaQuSo

- Introduction
- Package decomposition
- Coupling
- Dependency Structure
- Code duplication
- Malpractices
- Conclusions

Stability assessment

Package decomposition	Sotograph	good
Coupling	Sotograph	satisfactory
Architecture	SA4J	poor
Code duplication	IntelliJ IDEA	satisfactory
	Gemini	satisfactory
Malpractices	ESC/Java	poor

?

Stability assessment

"Bad code compromises good design"

- Design is quite satisfactory
- Implementation
  - Violates the design
    - package communication
    - architecture
  - Introduces malpractices
- Our analysis provided insight in development process
  - Emphasis on early stages of development (design)
  - Lack of time and resources during the implementation

Tools assessment

- Correct analysis requires tools ranging from design analysis to code analysis
  - Ideally also requirements analysis
  - Tooling is really valuable
- Tools' discoveries are consistent
- Effort
  - Application : Low
    - except for ESC/Java: High
  - Interpretation: Medium
    - except for SA4J: Low
    - except for ESC/Java: High

LaQuSo

Conclusions

- Stability can be operationalized in terms of tool-supported issues
- Measurements are clear, interpretation may be challenging
- Assertion checking provides new insights:
  - Proof complexity = code complexity
  - Failure to prove correctness may be caused by instability

LaQuSo
Laboratory for Quality Software

Visit Address:
TU/e campus, Hoofdgebouw 5.91
Den Dolech 2 Eindhoven

Mail Address:
HG 5.91
Postbus 513
5600 MB Eindhoven

Telephone:
040-2472526
Fax:
040-2474252

Email:
info@laquso.com
Web Site:
www.laquso.com

Thank you!

Any Questions?

Symposium VVSS2005 about
Verification & Validation of Software
24th of November in Eindhoven, NL
See: www.laquso.com

LaQuSo is an activity of Technische Universiteit Eindhoven