



SISSy – a Tool for Structural Investigation of Object-Oriented Software Systems

Mircea Trifu
Adrian Trifu

FZI Forschungszentrum Informatik
Software Engineering Group
10-14 Haid-und-Neu Street
76131 Karlsruhe

17.11.2006

Software quality



- Constructing high quality software is easier said than done:
 - 1968 - NATO proclaims the *software crisis*: software systems have bad product quality and cause unreasonably high maintenance costs!
 - 1994 - IBM Survey on large software projects with 24 IT companies: 88% of the software systems require a major redesign!
- External quality = customer's perspective: Quality of the product
- Internal quality = developer's perspective: Quality of the design
- Benefits of high internal quality
 - Lower development and maintenance costs.
 - Positive effect on external quality (fewer bugs, performance, stability)

Reasons for low internal quality

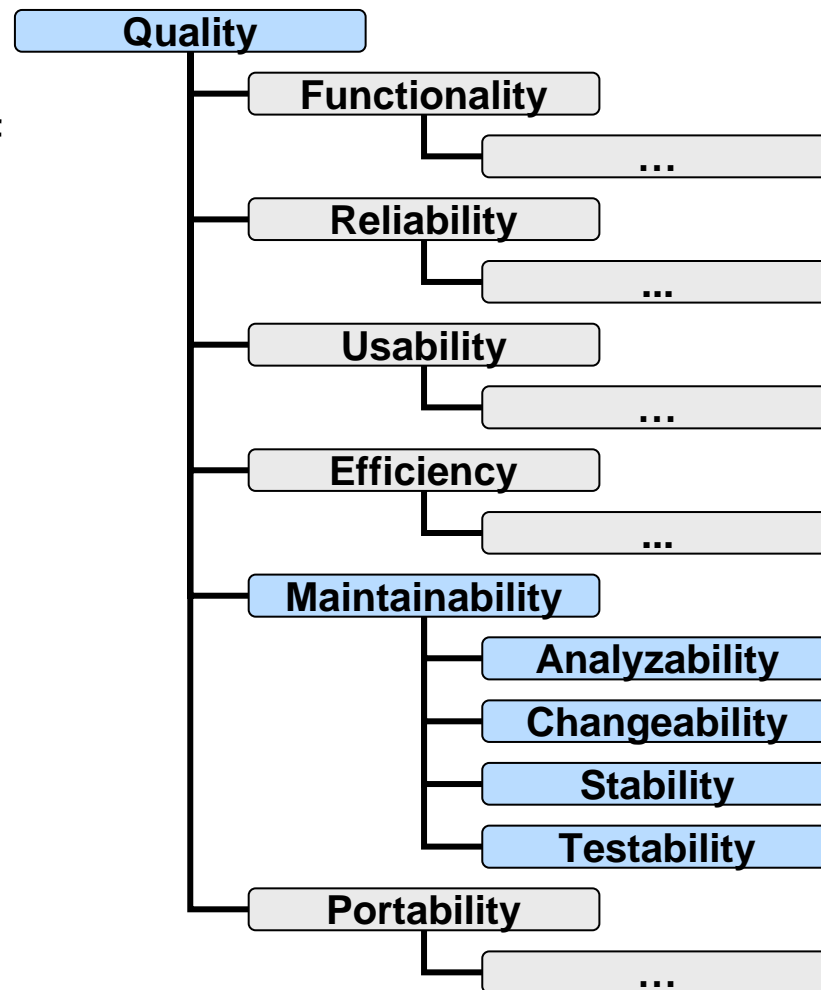


- Changing requirements
 - Unexpected extensions
 - New usage contexts
- Long lifespan of current software systems
 - Undocumented / misunderstood architecture
 - Faulty adaptations
- Time and cost pressure
- Lack of knowledge about good OO structures
- Human error

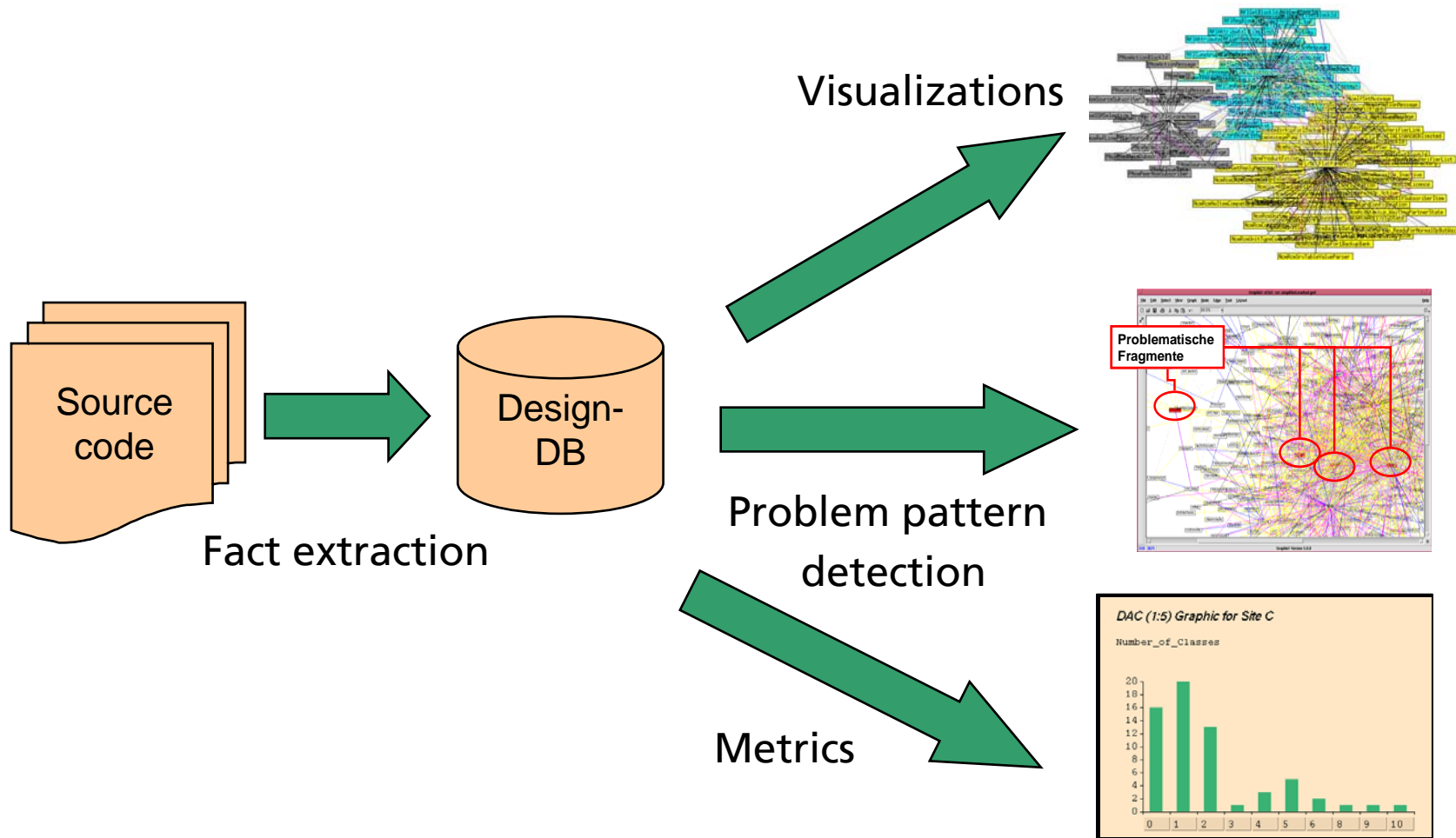
→ Software structures unavoidably degrade over time

Assessing internal software quality

- ISO 9126
 - Abstract decomposition of quality
- FCM models
 - ISO model + Metrics
- FS Model
 - ISO model + problem patterns
 - Suggests correction steps

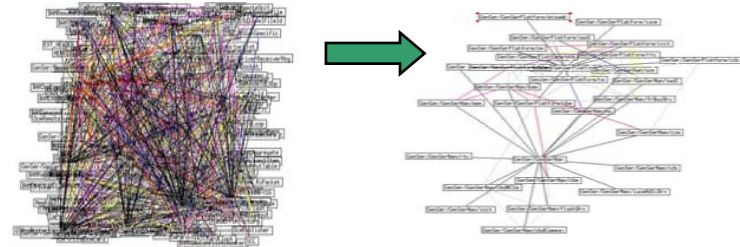


Overview of our techniques



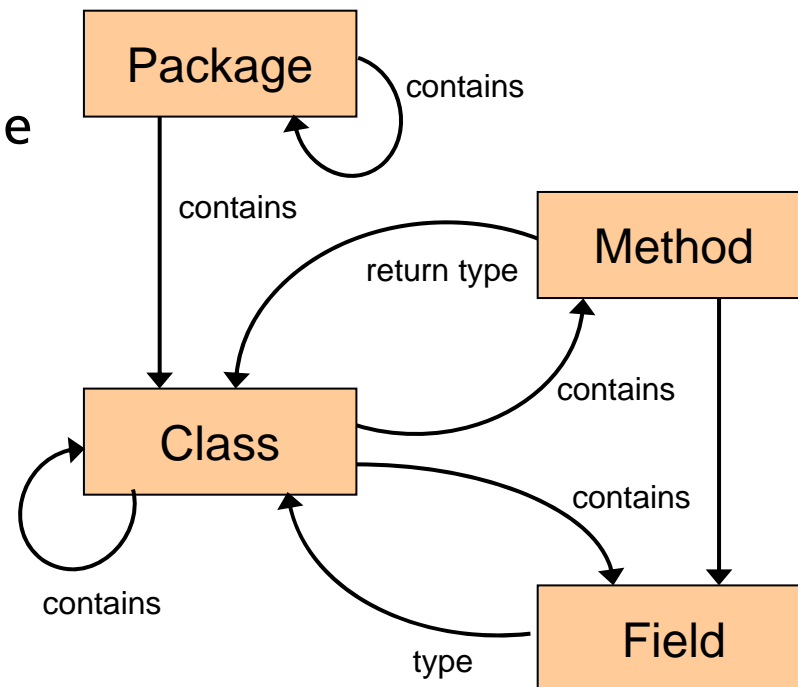
Fact extraction

- Objective
 - An abstract model of the system (design database) as a foundation for further analyses



- Techniques:
 - Compiler techniques
 - Graph theory to abstract the design database

- Problems:
 - Programming language issues (C/C++: macros!)
 - Incomplete or defective source code



Software metrics and problem patterns



- Software metrics
 - Count: lines of code, classes, methods, attributes
 - Calculate: depth of inheritance, cyclomatic complexity, coupling, cohesion
- Problem patterns
 - Architecture level: package and subsystem structure
 - Design level: inheritance hierarchies, class interaction and modularization
 - Implementation level: within a method body

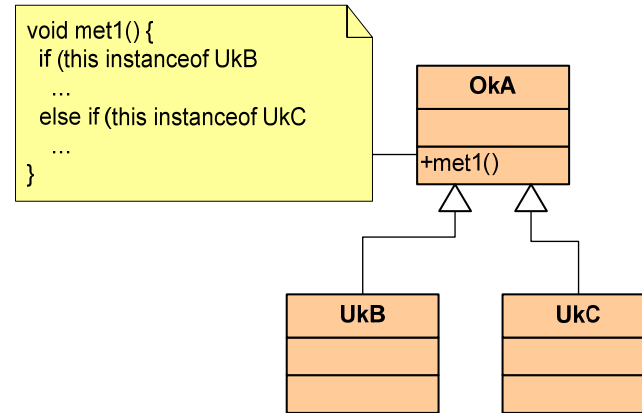
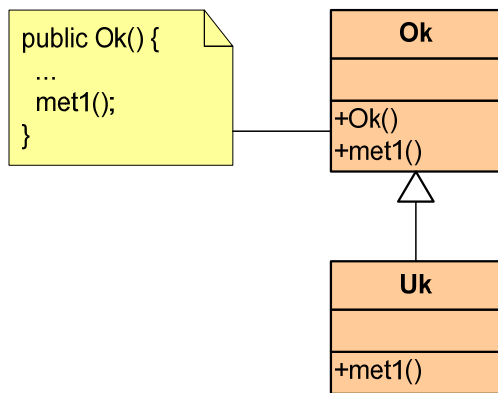
Our Arsenal



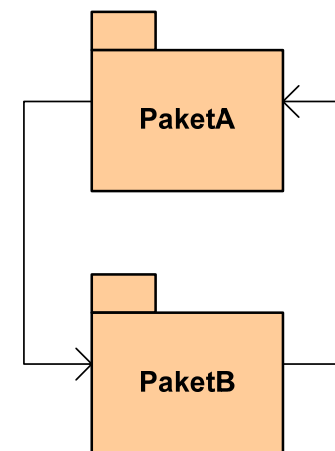
Architectural level	Design level	Implementation level
Cyclical dependency between packages Dead imports God package Import chaos Mini-packages Reversed package and inheritance hierarchies Type name duplication Unfinished code	Attribute overlap Constant redefinition Cyclical dependency between classes Dead attribute Dead method General parameter Generation conflict God class, attribute form God class, method form Ignored abstraction Inconsistent operations Interface bypass Knows of derived Late abstraction Mini-classes Orphan sibling attributes Orphan sibling methods Permissive visibility, attribute form Permissive visibility, method form Polymorphic calls in constructor Polymorphism placebo Refused bequest, implementation form Refused bequest, interface form Similar unrelated abstractions Simulated polymorphism Violation of data encapsulation	Complex method Cyclical dependency between source files Cyclical method calls Dead code Duplicated code God file God method Improper name length Inappropriate commenting Informal documentation Long parameter list Misleading naming of source files Object placebo, attribute form Object placebo, method form Overloaded file Risky code Variables having constant value Violation of naming convention

Problem examples

- Simulated Polymorphism



- Polymorphic calls in constructors



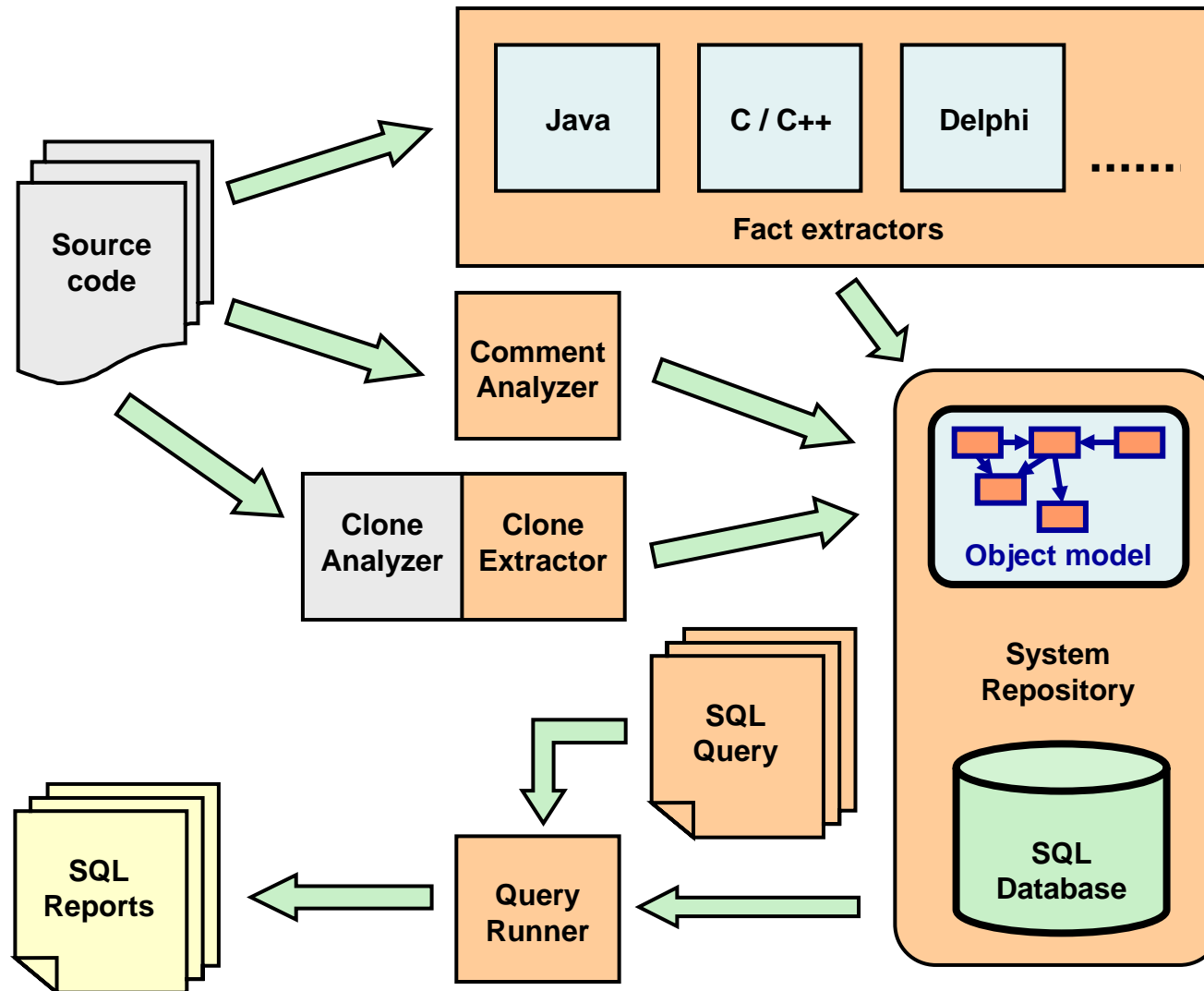
- Cyclical dependencies between packages

Problem examples II



- Code clones
 - Implementation level: Duplicated code
 - Design level: Similar unrelated abstractions
- High complexity
 - Complex method
 - Long parameter list
- Dead code
 - Dead method
 - Dead attribute
 - Dead import

Our tool chain SISSy (<http://sissy.fzi.de>)



- **Single assessment**
 - A snapshot of the current state
 - External specialists may be involved
 - Assess a new release or a newly acquired code base
- **Build process integration**
 - Code is automatically inspected on check-in
 - Direct and timely feedback to the appropriate developers
- **Periodic trend analysis for project management**
 - Comparison with previous releases
 - Early identification of potential maintenance risks
 - Custom quality model, tailored on specific needs of the enterprise

Thank you!

Mircea Trifu (mircea.trifu@fzi.de)

Adrian Trifu (adrian.trifu@fzi.de)

FZI Forschungszentrum Informatik

Software Engineering Group

10-14 Haid-und-Neu Street

76131 Karlsruhe

Web: <http://www.fzi.de/prost>

Tools: <http://sissy.fzi.de>