Forschungszentrum Informatik

an der Universität Karlsruhe

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)

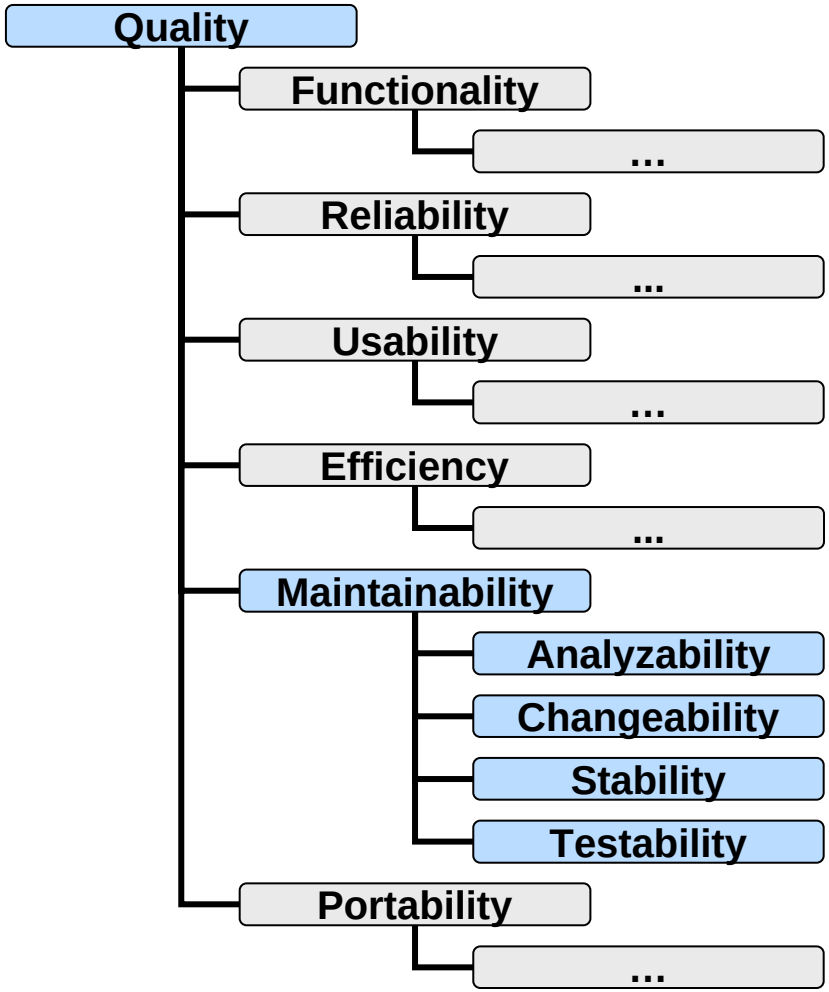
Reasons for low internal quality

- 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

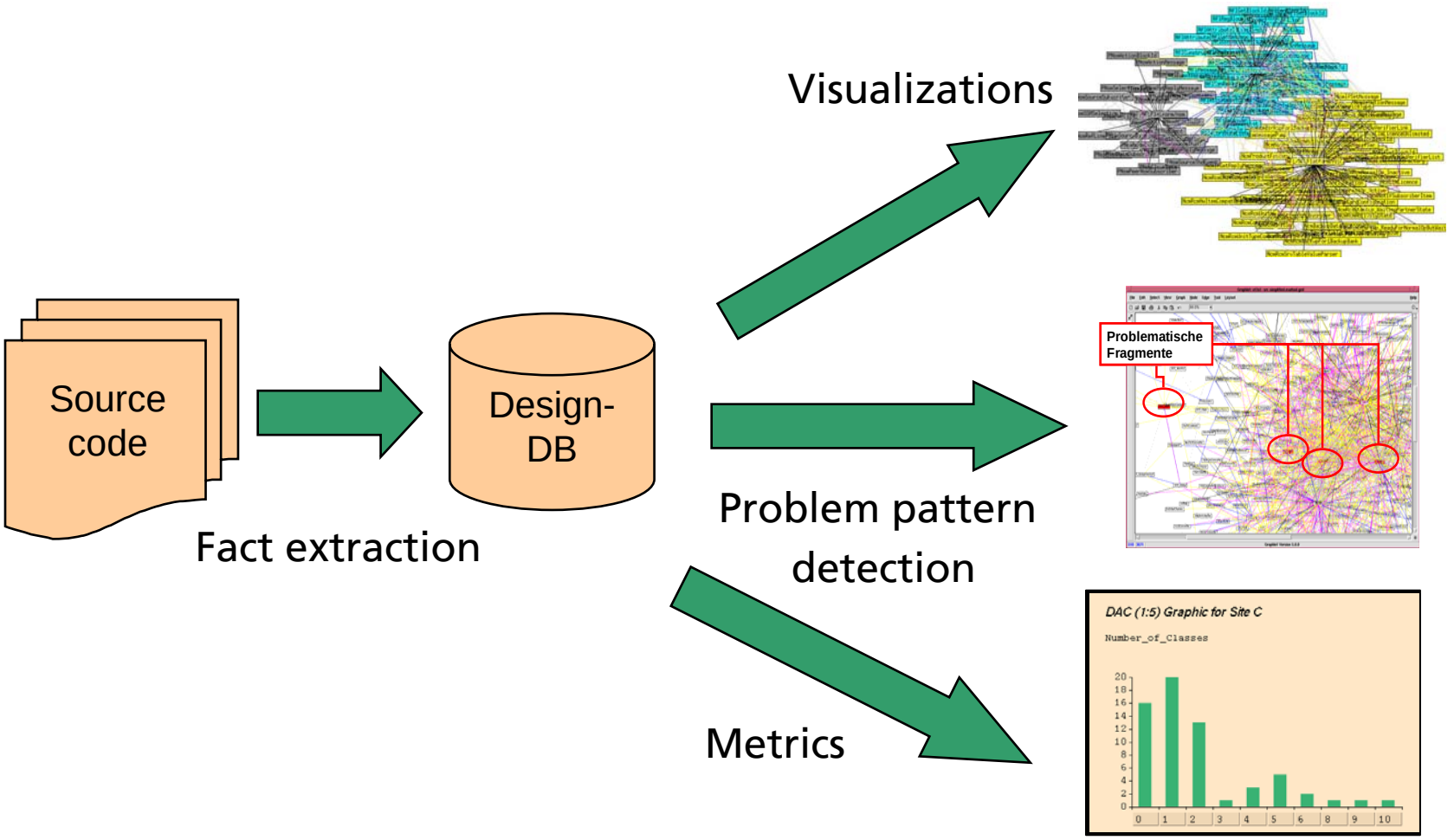
→ Software structures unavoidably degrade over time

Assessing internal software quality

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



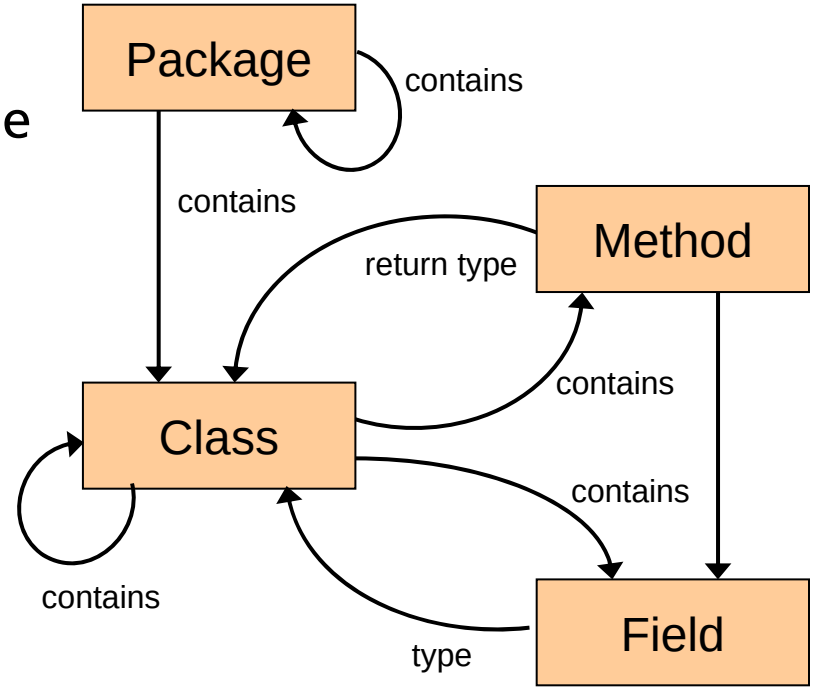
Overview of our techniques



Fact extraction

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

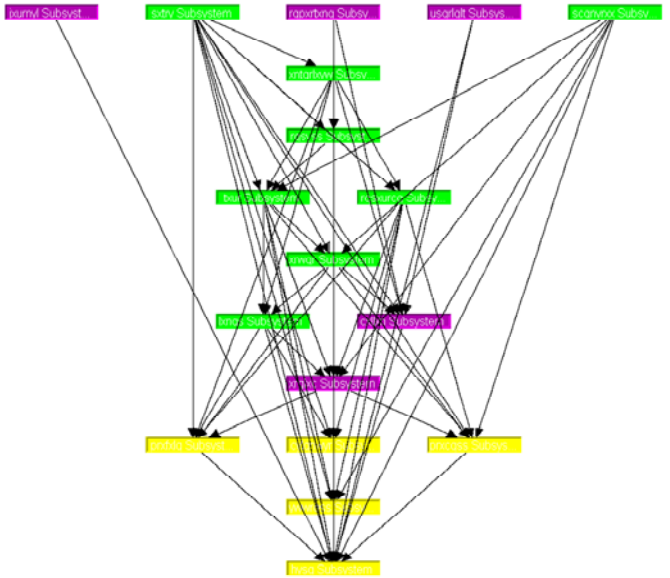
- Techniques:
  - Compiler techniques
  - Graph theory to abstract the design database
- Problems:
  - Programming language issues (C/C++: macros!)
  - Incomplete or defective source code

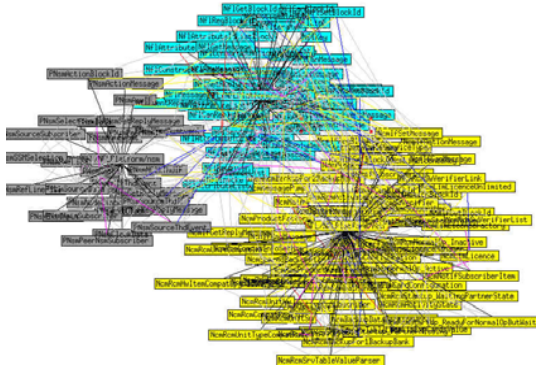


Visualization

- Objective:
  Use the fact that a picture may say more than a 1000 words! Visualizations help to grasp complex structures.

- Usage Scenarios:
  - Understand architectures and structures
  - Check dependencies (layered architecture, framework vs. application code)





Software metrics and problem patterns

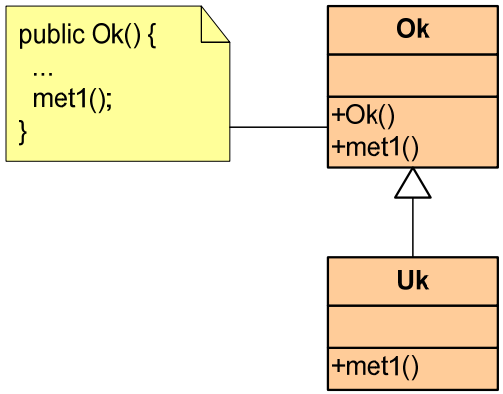
- Software metrics
  - Count: lines of code, classes, methods, attributes
  - Calculate: depth of inheritance, cyclomatic complexity, coupling, cohesion
- Problem patterns
  - Architecture level: package and subsystem structure
  - Design level: inheritance hierarchies, class interaction and modularization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

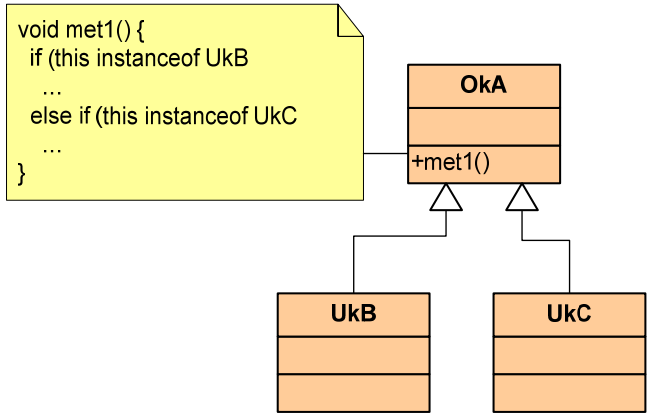
Our Arsenal

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

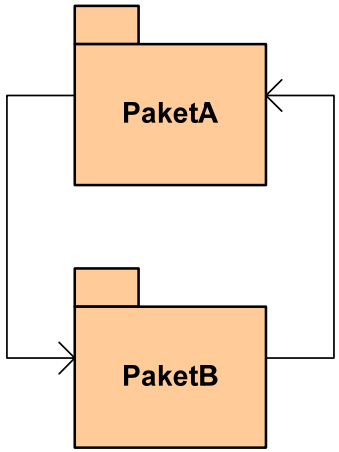
Problem examples

- Simulated Polymorphism





- Polymorphic calls in constructors

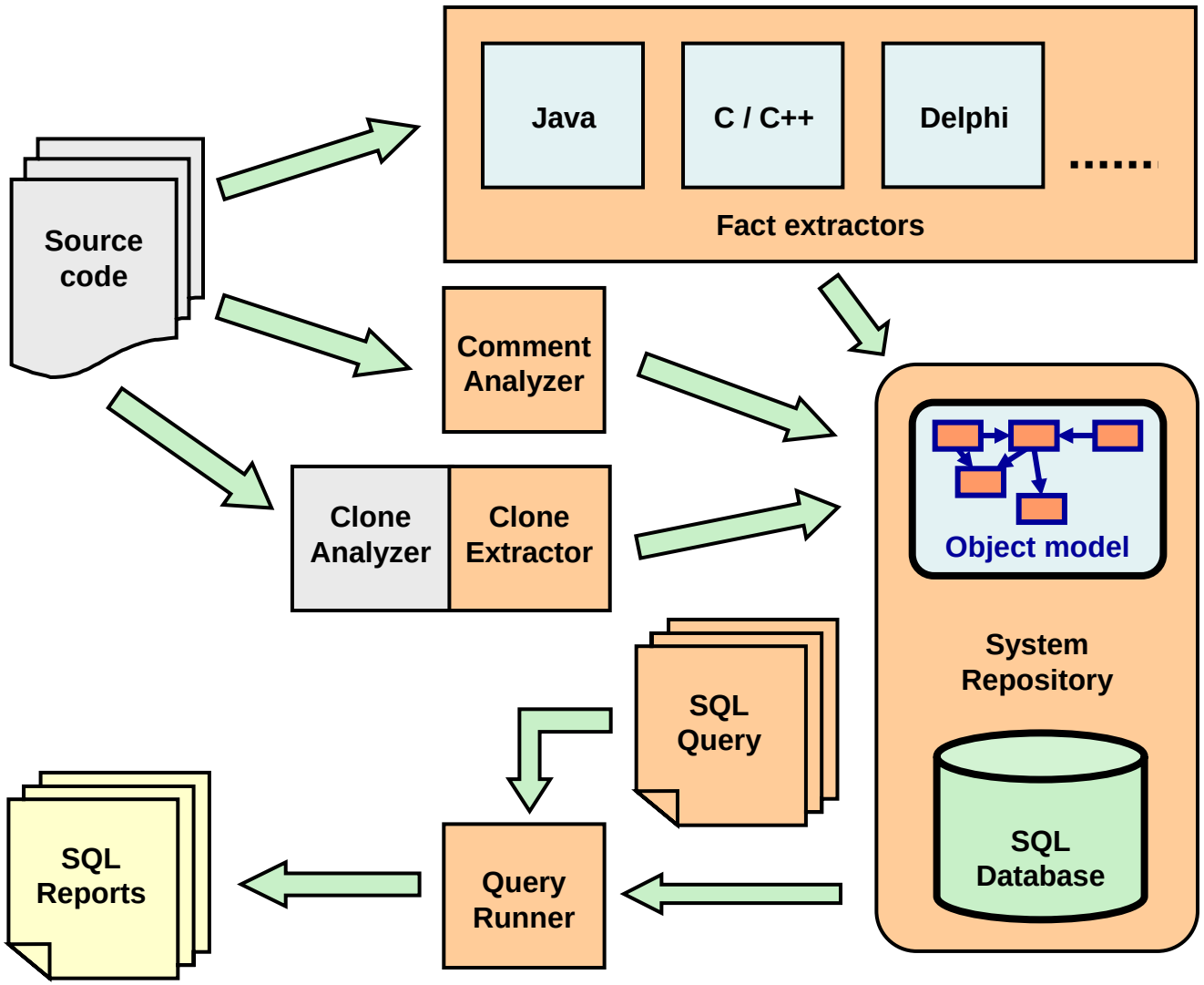


- Cyclical dependencies between packages

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)



Use cases

- 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