Software Improvement Group

Software Analysis and Testing

Métodos Formais em Engenharia de *Software*

January 2009
Joost Visser

Arent Janszoon Ernststraat 595-H
NL-1082 LD Amsterdam
info@sig.nl
www.sig.nl

Me

Software Improvement Group

CV

- Technical University of Delft, Computer Science, MSc 1997
- University of Leiden, Philosophy, MA 1997
- CWI (Center for Mathematics and Informatics), PhD 2003
- Software Improvement Group, developer, consultant, etc, 2002-2003
- Universidade do Minho, Post-doc, 2004-2007
- Software Improvement Group, head R&D, 2007-…

Research

- Grammars, traversal, transformation, generation
- Functional programming, rewriting strategies
- Software quality, metrics, reverse engineering

Software Improvement Group

Company

- Spin-off from CWI in 2000, self-owned, independent
- Management consultancy grounded in source code analysis
- Winner of the Innovator Award 2007

Services

- Software Risk Assessments (snapshot) and Software Monitoring (continuous)
- Toolset enables to analyze source code in an automated manner
- Experienced staff transforms analysis data into recommendations
- We analyze over 50 systems annually
- Focus on technical quality, primarily maintainability / evolvability

Who is using our services?

Software Improvement Group

Financials / Insurance companies

ABN·AMRO
PGGM
ING
Zorg en Zekerheid
Rabobank
InterBank
KREDIET WAAR JE LANGER MEE GENIET
achmea
ZwitserLeven
Friesland Bank
delta lloyd
LeasePlan
Allianz

Government

RVR
Raden Voor Rechtsbijstand
VROM
RDW
POLITIE
cwi
Ministerie van Binnenlandse Zaken en Koninkrijksrelaties
SVB
kadaster

Logistical

KLM
DHL
TNT
EUROMAXX TERMINAL
norfolkline
DOVER - DUNKERQUE FERRIES
ProRail

IT

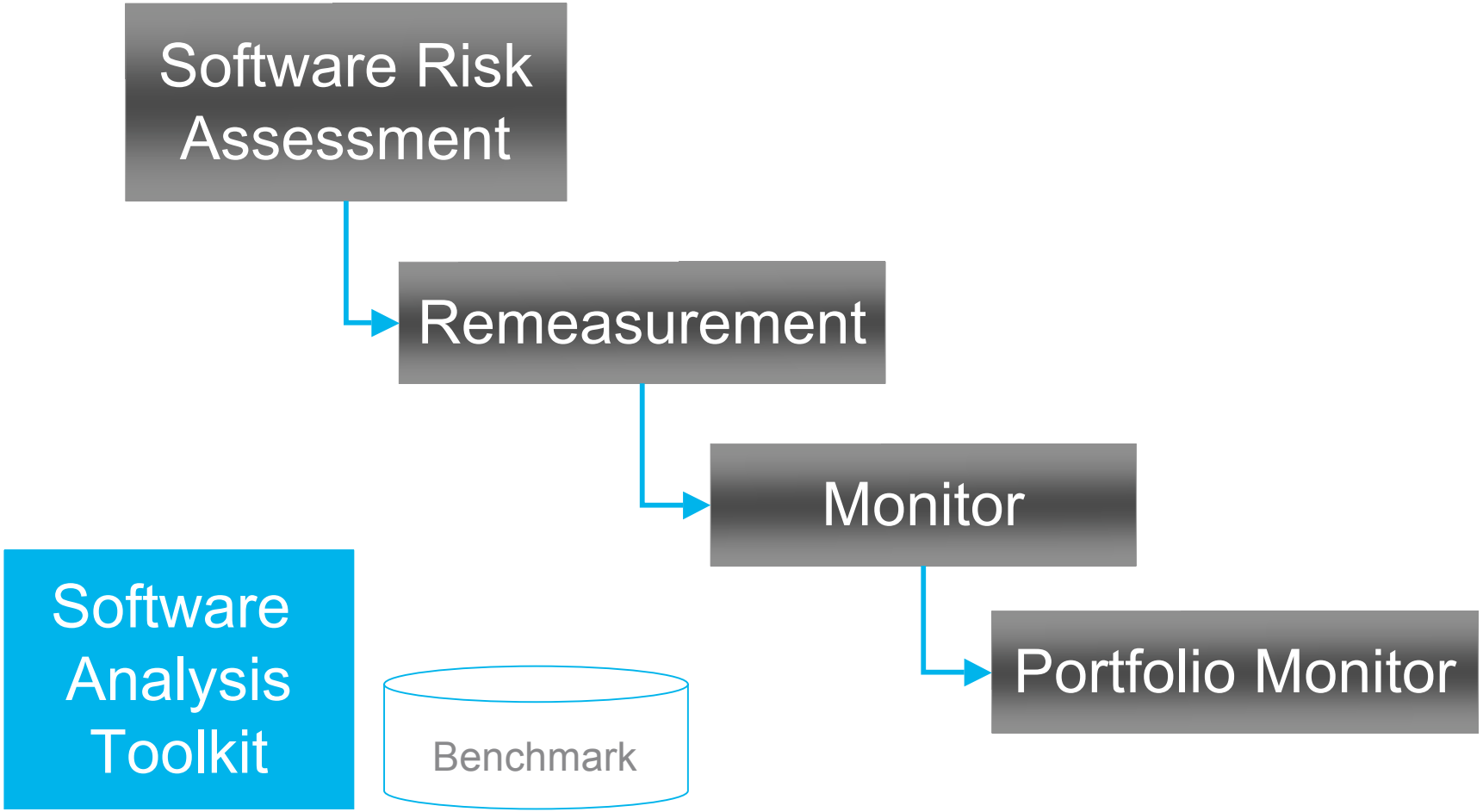
Getronics PinkRoccade
CENTRIC
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Exact software
IBM
CHESS
logicaCMG

Other

ENECO energie
PRICEWATERHOUSECOOPERS
CARe
SCHADESERVICE
KPMG
oxxio
Gasunie

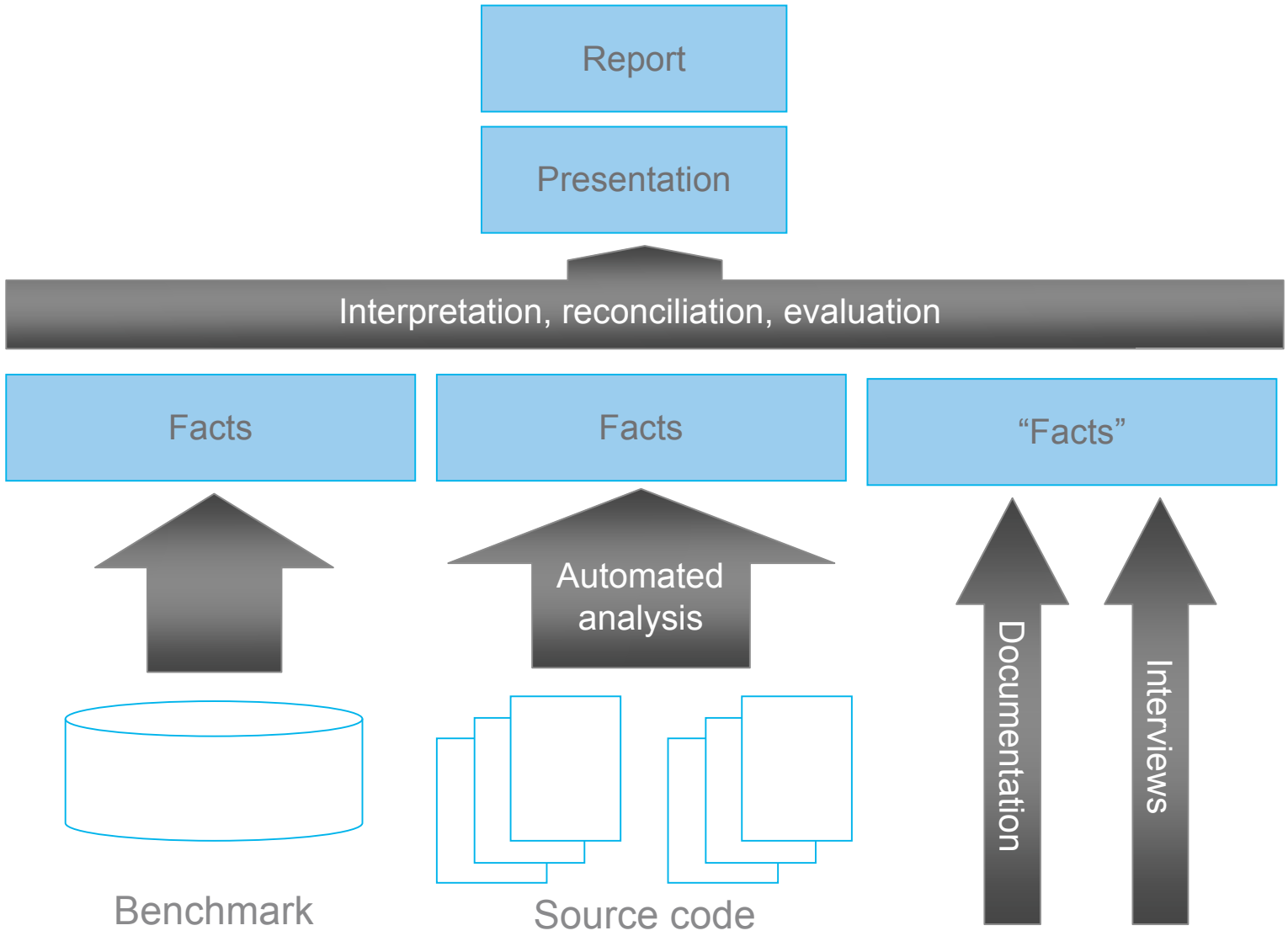
Our services

Software Improvement Group



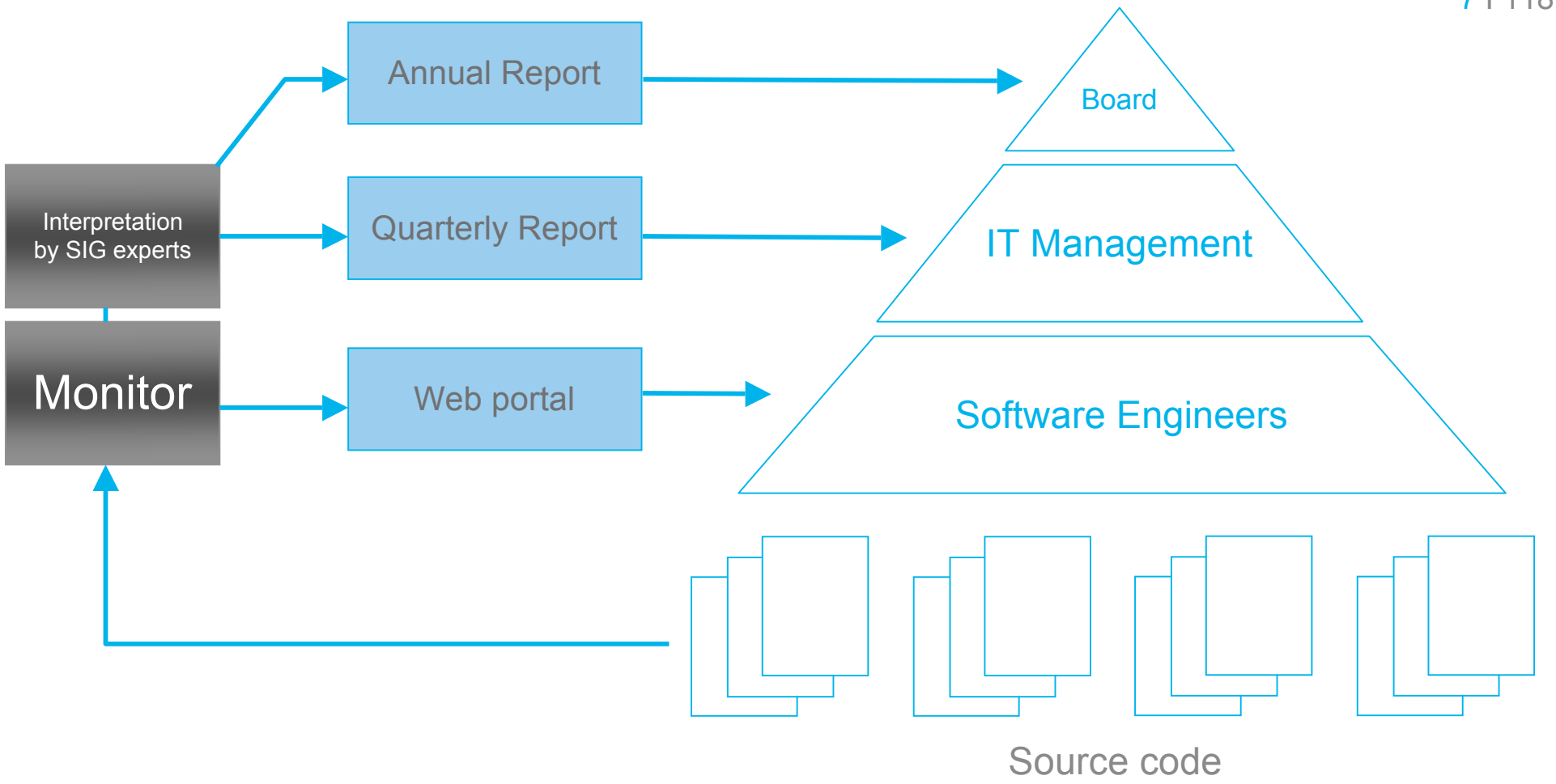
Software Risk Assessment

Software Improvement Group



Software Quality Monitor

Software Improvement Group



Structure of the lecture

- Introduction SIG

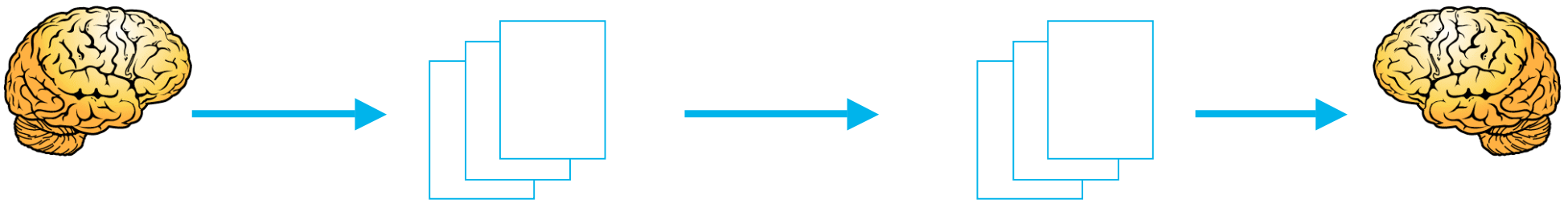
- General overview of software analysis and testing
- Testing
- Patterns
- Quality & metrics
- Reverse engineering

Create | **Change** | **Analyze**

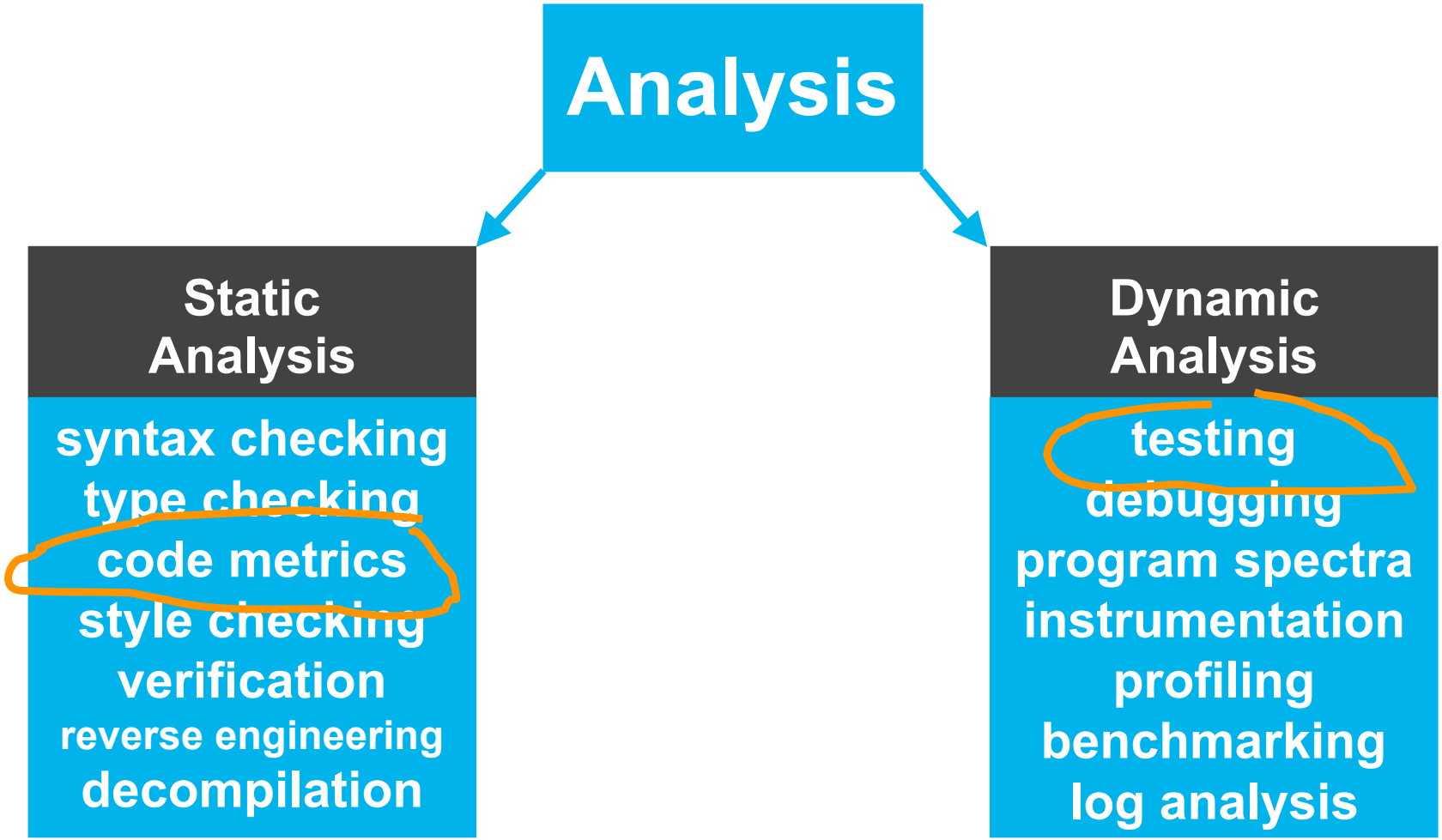
requirements analysis
design, code, compile
configure, install

refactor, fix, patch
maintain, renovate
evolve, update, improve

understand, assess
evaluate, test
measure, audit



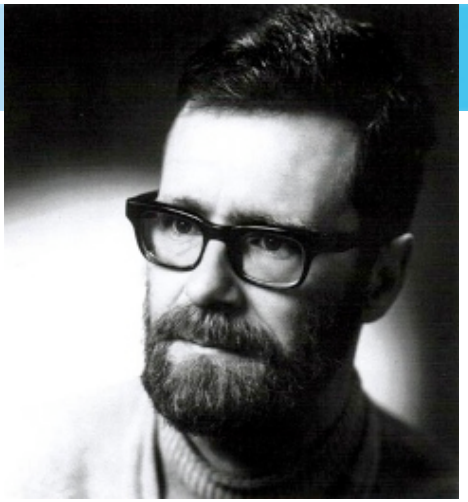
Software Improvement Group



Is testing un-cool?

Software Improvement Group

Edsger Wybe Dijkstra (1930 - 2002)



- "Program testing can be used to show the presence of bugs, but never to show their absence!"
  Notes On Structured Programming, 1970

- "Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence."
  The Humble Programmer, ACM Turing Award Lecture, 1972

Does not mean: "Don't test!!"

Is testing un-cool?

Software Improvement Group

Industry

- Testers earn less then developers
- Testing is “mechanical”, developing is “creative”
- Testing is done with what remains of the budget in what remains of the time

Academia

- Testing is not part of the curriculum, or very minor part
- Verification is superior to testing
- Verification is more challenging than testing

Software Analysis. How much?

Software Improvement Group

50 - 75%

In a typical commercial development organization, the cost of providing [the assurance that the program will perform satisfactorily in terms of its functional and nonfunctional specifications within the expected deployment environments] via appropriate debugging, testing, and verification activities can easily range from 50 to 75 percent of the total development cost. (Hailpern and Santhanam, 2002)

Software Analysis. Enough?

Software Improvement Group

Planning Report 02-3
The Economic Impacts of Inadequate Infrastructure for Software Testing

$$\$60 \times 10^9$$

Table ES-4. Costs of Inadequate Software Testing Infrastructure on the National Economy

	The Cost of Inadequate Software Testing Infrastructure (billions)	Potential Cost Reduction from Feasible Infrastructure Improvements (billions)
Software developers	$21.2	$10.6
Software users	$38.3	$11.7
Total	$59.5	$22.2

of total impacts, and software users accounted for the about 60 percent.

Software Analysis. More?

Software Improvement Group

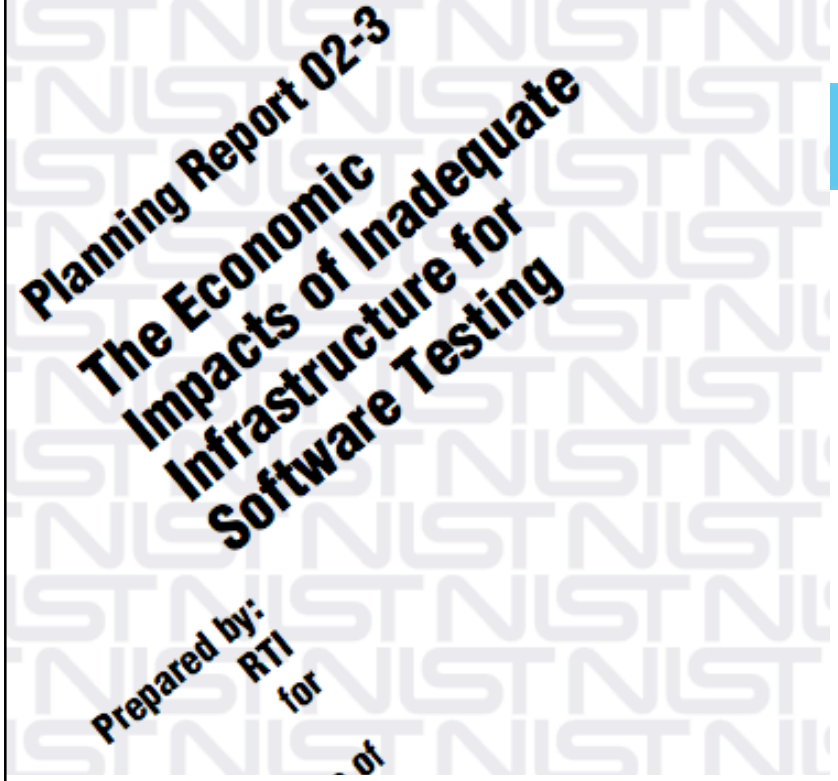
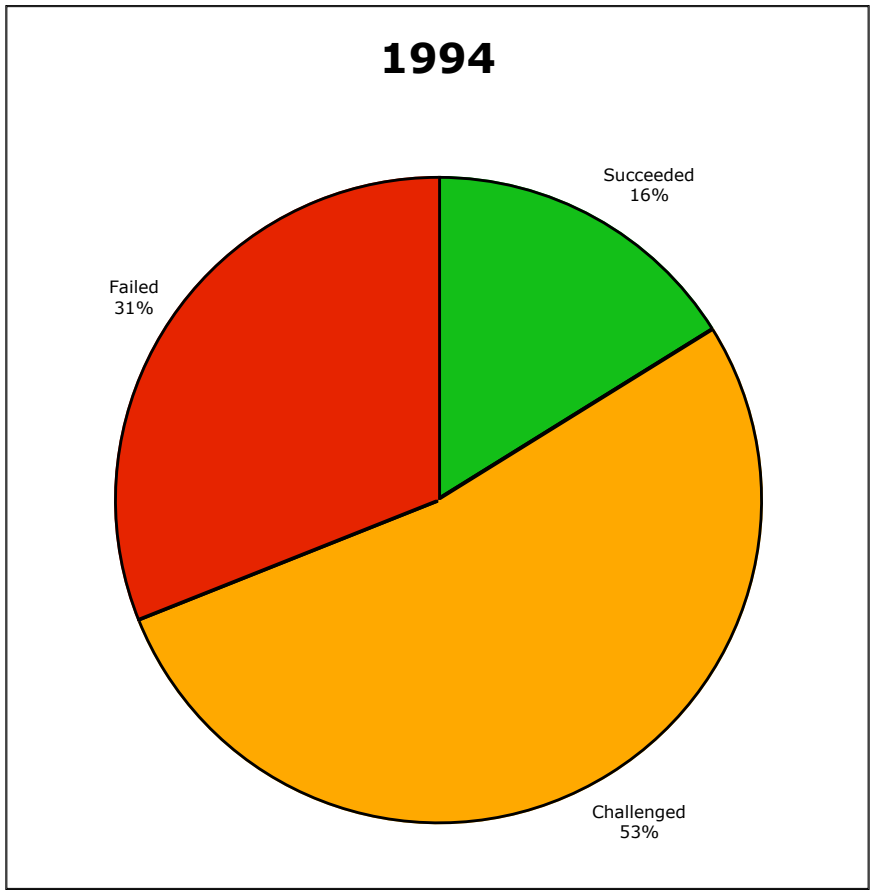


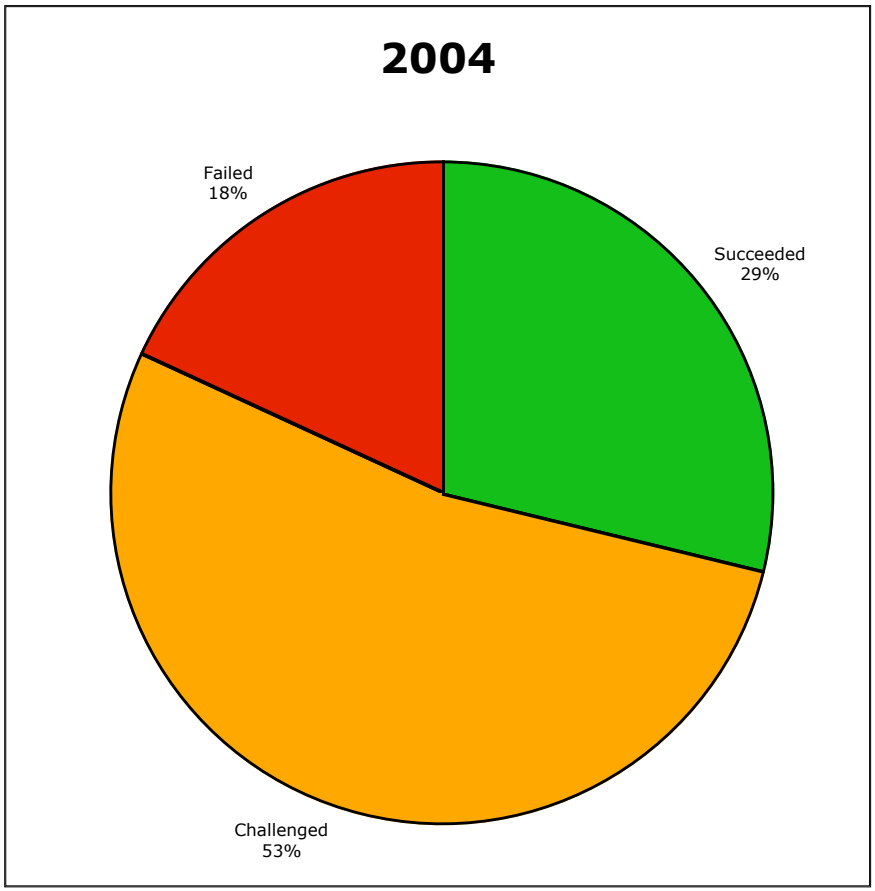
Table 1-4. Recent Aerospace Losses due to Software Failures

	Airbus A320 (1993)	Ariane 5 Galileo Poseidon Flight 965 (1996)	Lewis Pathfinder USAF Step (1997)	Zenit 2 Delta 3 Near (1998)	DS-1 Orion 3 Galileo Titan 4B (1999)
Aggregate cost		$640 million	$116.8 million	$255 million	$1.6 billion
Loss of life	3	160			
Loss of data		Yes	Yes	Yes	Yes

Software Analysis
Room for improvement?

Software Improvement Group





Standish Group, *"The CHAOS Report"*

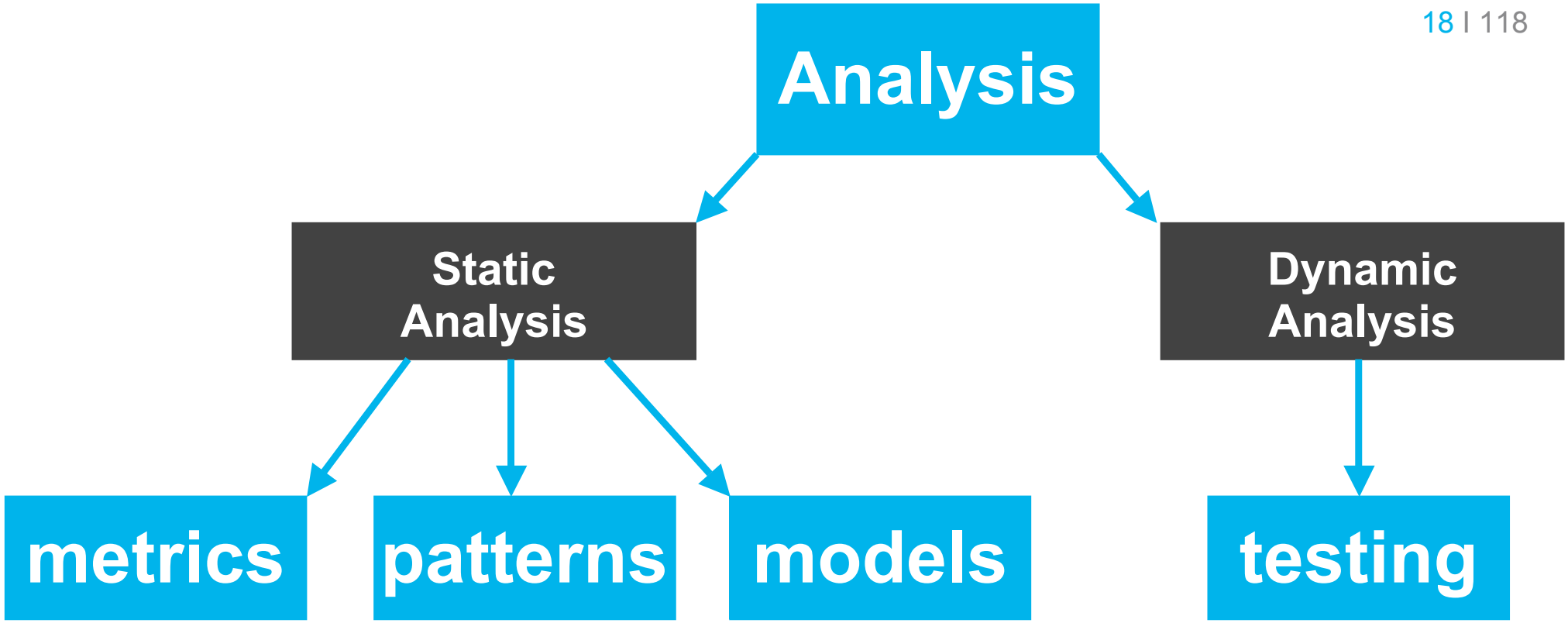
So

- Testing ⊂ Dynamic analysis ⊂ Analysis ⊂ S.E.
- Analysis is a major and essential part of software engineering
- Inadequate analysis costs billions

- More effective and more efficient methods are needed
- Interest will keep growing in both industry and research

Structure of the lectures

Software Improvement Group



TESTING

Testing

Kinds

- Conformance
- Interoperability
- Performance
- Functional
- White-box
- Black-box
- Acceptance
- Integration
- Unit
- Component
- System
- Smoke
- Stress

Ways

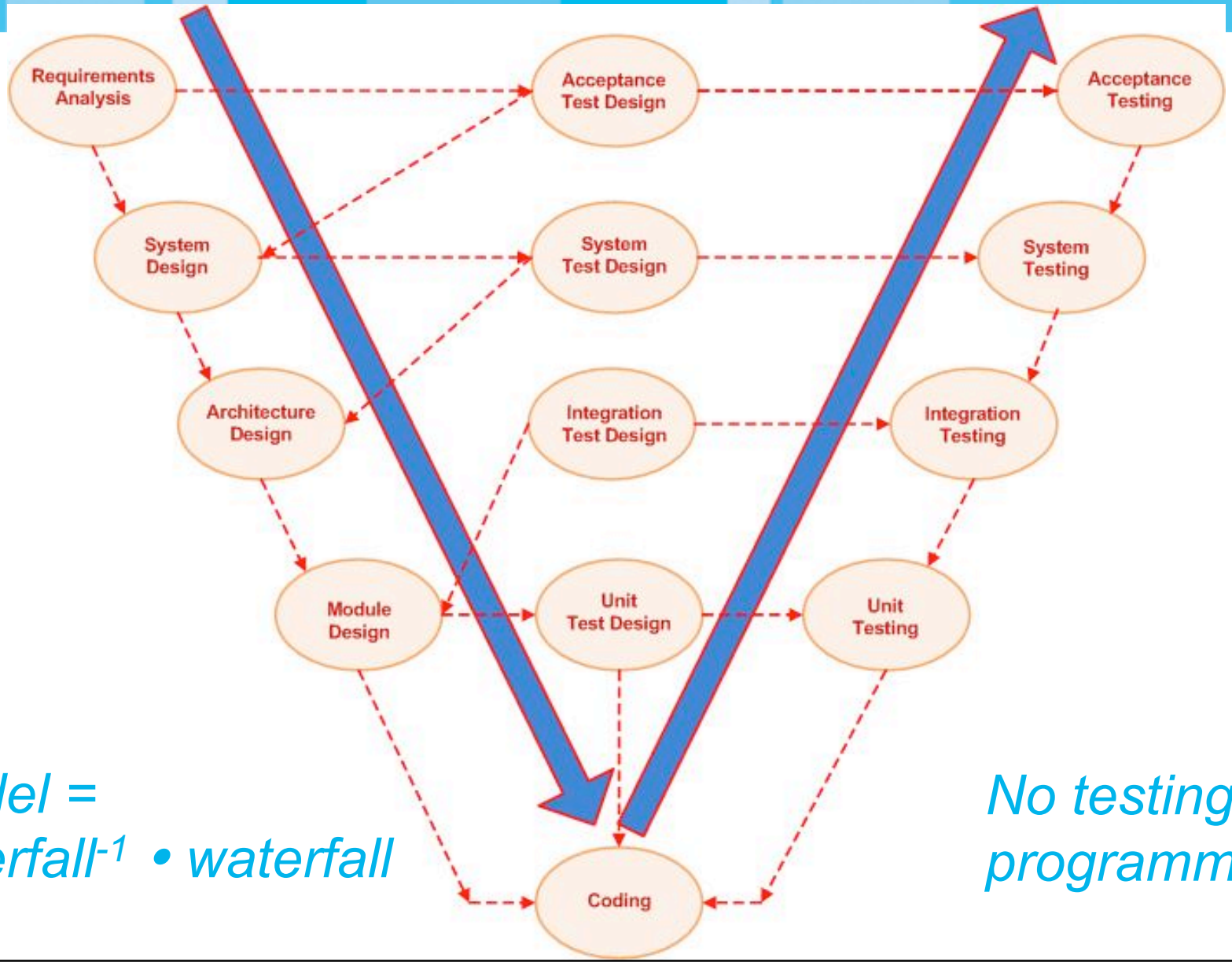
- Manual
- Automated
- Randomized
- Independent
- User
- Developer

With

- Plans
- Harness
- Data
- Method
- Frameworks

Testing
V-model

Software Improvement Group



V-model = waterfall^{-1} • waterfall

No testing while programming!

Testing
Eliminate waste

Software Improvement Group

Waste

- Coding and debugging go hand-in-hand
- Coding effort materializes in the delivered program
- Debugging effort? Evaporates!

Automated tests

- Small programs that capture debugging effort.
- Invested effort is consolidated …
- … and can be re-used without effort ad-infinitum

Unit testing

What is unit testing?

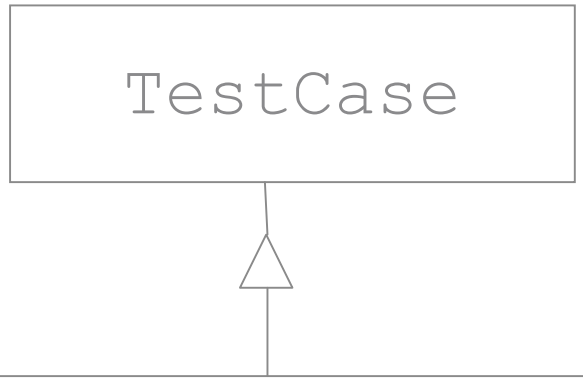
Software Improvement Group

A unit test is …

- fully automated and repeatable
- easy to write and maintain
- non-intrusive
- documenting
- applies to the simplest piece of software

Tool support

- JUnit and friends




```
public void testMyMethod {
   X x = …;
   Y y = myMethod(x);
   Y yy = …;
   assertEquals("WRONG",yy,y)
}
```

Testing goals

Unit testing has the following goals:

- Improve quality
  - Test as specification
  - Test as bug repellent
  - Test as defect localization
- Help to understand
  - Test as documentation
- Reduce risk
  - Test as a safety net
  - Remove fear of change

Observing unit-testing maturity in the wild (characterization of the population)

Software Improvement Group

Organization

- public, financial, logistics
- under contract, in house, product software
- with test departments, without test departments

Architecture & Process

- under architecture, using software factories
- model driven, handwritten
- open source frameworks, other frameworks
- using use-cases/requirements
- with blackbox tools, t-map

Technology

- information systems, embedded
- webbased, desktop apps
- java, c#, 4GL's, legacy
- latest trend: in-code asserts (java.spring)

Stage 1
No unit testing

Software Improvement Group

Observations:

- Very few organizations use unit testing
- Also brand new OO systems without any unit tests
- Small software shops and internal IT departments
- In legacy environments: programmers describe in words what tests they have done.

Symptoms:

- Code is instable and error-prone
- Lots of effort in post-development testing phases

Stage 1
No unit testing

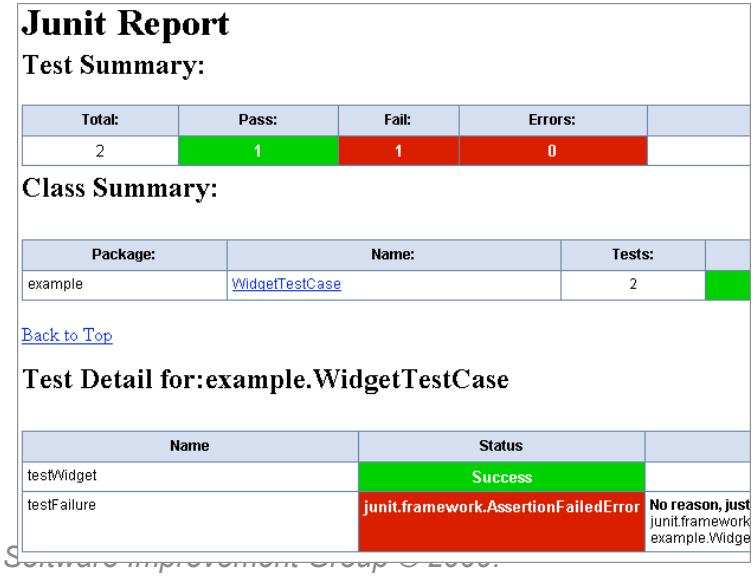
Software Improvement Group

Excuses:

- “It is just additional code to maintain”
- “The code is changing too much”
- “We have a testing department”
- “Testing can never prove the absence of errors”
- “Testing is too expensive, the customer does not want to pay for it”
- “We have black-box testing”

Action

- Provide standardized framework to lower threshold
- Pay for unit tests as deliverable, not as effort



Junit Report

Test Summary:

Total:	Pass:	Fail:	Errors:	
2	1	1	0	

Class Summary:

Package:	Name:	Tests:	
example	WidgetTestCase	2	

Back to Top

Test Detail for:example.WidgetTestCase

Name	Status	
testWidget	Success	
testFailure	junit.framework.AssertionFailedError	No reason, just junit.framework example.Widge

Stage 2
Unit test but no coverage measurement

Software Improvement Group

Observations

- Contract requires unit testing, not enforced
- Revealed during conflicts
- Unit testing receives low priority
- Developers relapse into debugging practices without unit testing
- Good initial intentions, bad execution
- Large service providers

Symptoms:

- Some unit tests available
- Excluded from daily build
- No indication when unit testing is sufficient
- Producing unit test is an option, not a requirement

Stage 2
Unit test but no coverage measurement

Software Improvement Group

Excuses:

- “There is no time, we are under pressure”
- “We are constantly stopped to fix bugs”

Actions

- Start measuring coverage
- Include coverage measurement into nightly build
- Include coverage result reports into process

Stage 3
Coverage, not approaching 100%

Software Improvement Group

Observations

- Coverage is measured but gets stuck at 20%-50%
- Ambitious teams, lacking experience
- Code is not structured to be easily unit-testable

Symptoms:

- Complex code in GUI layer
- Libraries in daily build, custom code not in daily build

Stage 3
Coverage, not approaching 100%

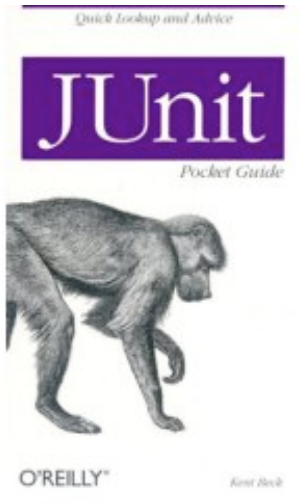
Software Improvement Group

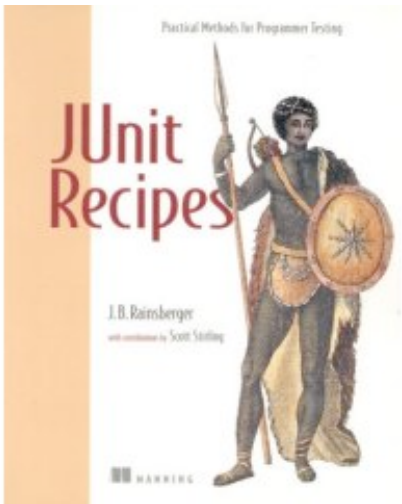
Excuses

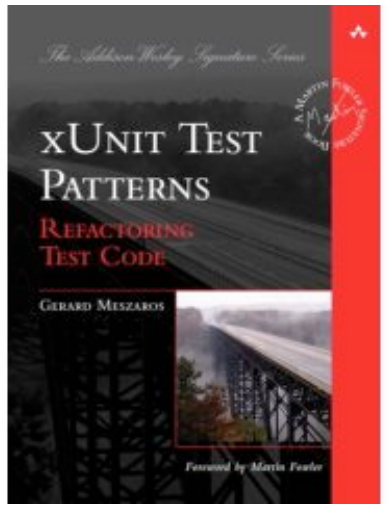
- “we test our libraries thoroughly, that affects more customers”

Actions:

- Refactor code to make it more easily testable
- Teach advance unit testing patterns
- Invest in set-up and mock-up







Stage 4
Approaching 100%, but no test quality

Observations

- Formal compliance with contract
- Gaming the metrics
- Off-shored, certified, bureaucratic software factories

Symptoms:

- Empty tests
- Tests without asserts.
- Tests on high-level methods, rather than basic units

- Need unit tests to test unit tests

Stage 4
Approaching 100%, but no test quality

Anecdotes:

- Tell me how you measure me, and I tell you how I behave
- We have generated our unit tests (at first this seems a stupid idea)

Action:

- Measure test quality
- Number of asserts per unit test
- Number of statements tested per unit test
- Ratio of number of execution paths versus number of tests

Stage 5
Measuring test quality

Software Improvement Group

Enlightenment:

- Only one organization: a Swiss company
- Measure:
  - Production code incorporated in tests
  - number of assert and fail statements
  - low complexity (not too many ifs)
- The process
  - part of daily build
  - “stop the line process”, fix bugs first by adding more tests
  - happy path and exceptions
  - code first, test first, either way

Testing
Intermediate conclusion

Software Improvement Group

Enormous potential for improvement:

- Do unit testing
- Measure coverage
- Measure test quality

- May not help Ariane 5
- Does increase success ratio for “normal” projects

Randomized Testing (quickcheck)

Randomized testing:

- QuickCheck: initially developed for Haskell
- Parameterize tests in the test data
- Property = parameterized test
- Generate test data randomly
- Test each property in 100 different ways each time

Test generation

Model-driven testing

Fault-injection

```
-- | Range of inverse is domain.
prop_RngInvDom r
  = rng (inv r) == dom r
    where
      types = r::Rel Int Integer
```

Is testing un-cool?

Software Improvement Group

Edsger Wybe Dijkstra (1930 - 2002)

- “Program testing can be used to show the presence of bugs, but never to show their absence!”

Martin Fowler

- “Don’t let the fear that testing can’t catch all bugs stop you from writing the tests that will catch most bugs.”

Simple test metrics

Line coverage

- Nr of test lines / nr of tested lines

Decision coverage

- Nr of test methods / Sum of McCabe complexity index

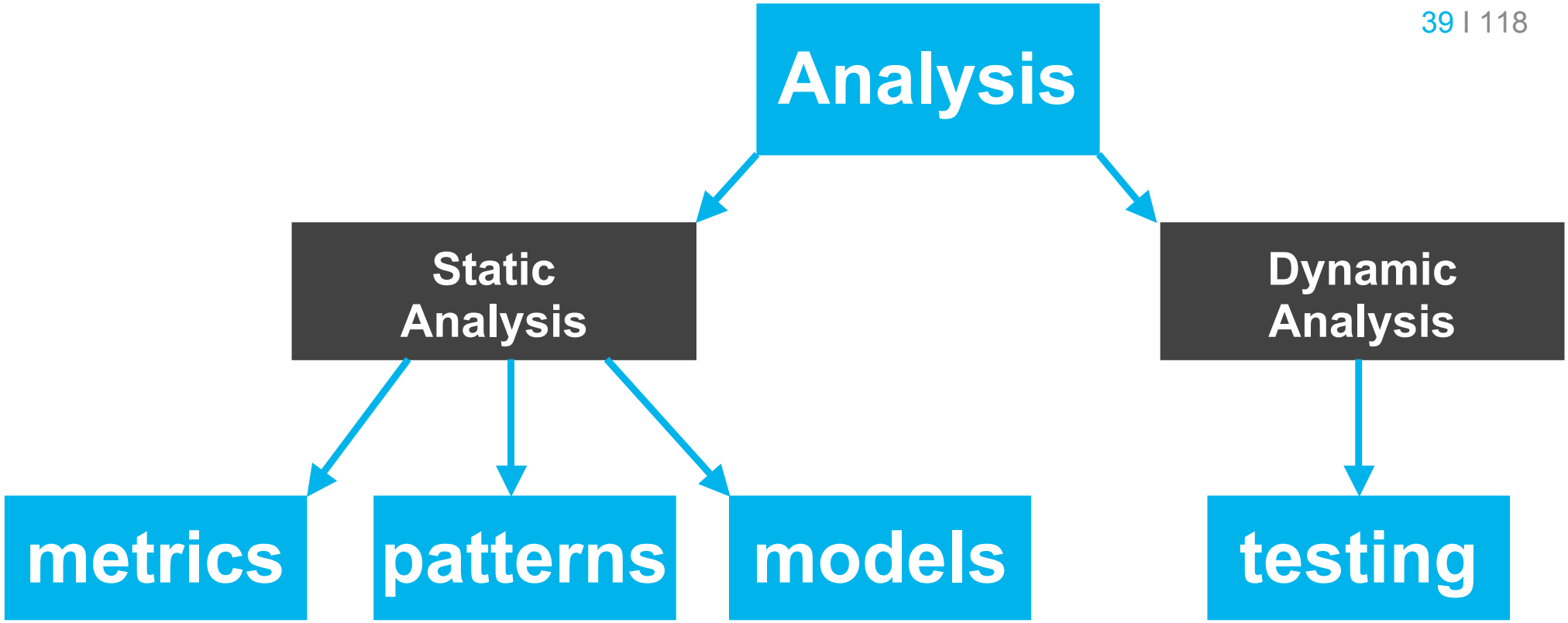
Test granularity

- Nr of test lines / nr of tests

Test efficiency

- Decision coverage / line coverage

Structure of the lecture



PATTERNS

Patterns

Coding style and coding standards

- E.g. layout, identifiers, method length, …

Secure coding guidelines

- E.g. SQL injection, stack trace visibility

Bug patterns

- E.g. null pointer dereferencing, bounds checking

Code smells

- E.g. “god class”, “greedy class”, ..

Checking coding style and coding standards

- Layout rules (boring)
- Identifier conventions
- Length of methods
- Depth of conditionals

Aim

- Consistency across different developers
- Ensure maintainability

Tools

- E.g. CheckStyle, PMD, …
- Integrated into IDE, into nightly build
- Can be customized

Software Improvement Group

Checking secure coding guidelines

- SQL injection attack
- Storing and sending passwords
- Stack-trace leaking
- Cross-site scripting

Aim

- Ensure security
- Security = Confidentiality + Integrity + Availability

Tools

- E.g. Fortify, Coverity

Detecting bug patterns

- Null-dereferencing
- Lack of array bounds checking
- Buffer overflow

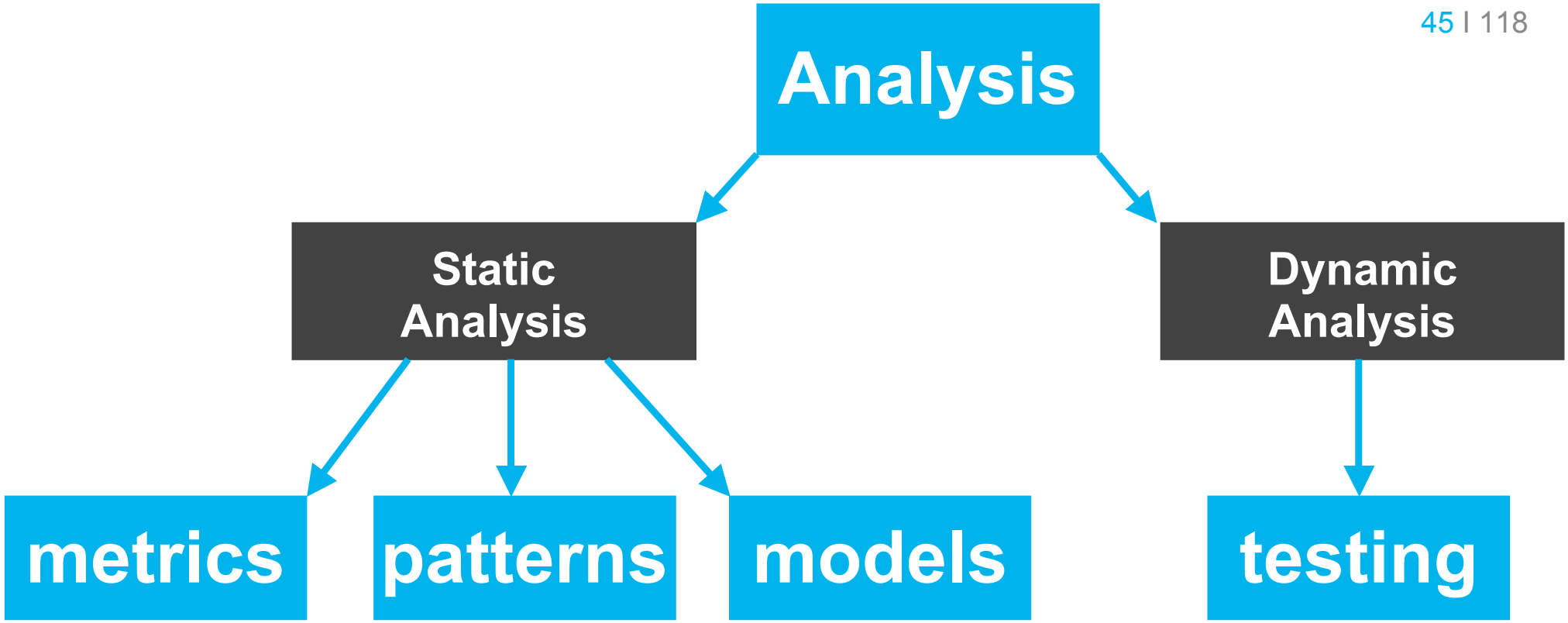
Aim

- Correctness
- Compensate for weak type checks

Tools:

- e.g. FindBugs
- Esp. for C, C++

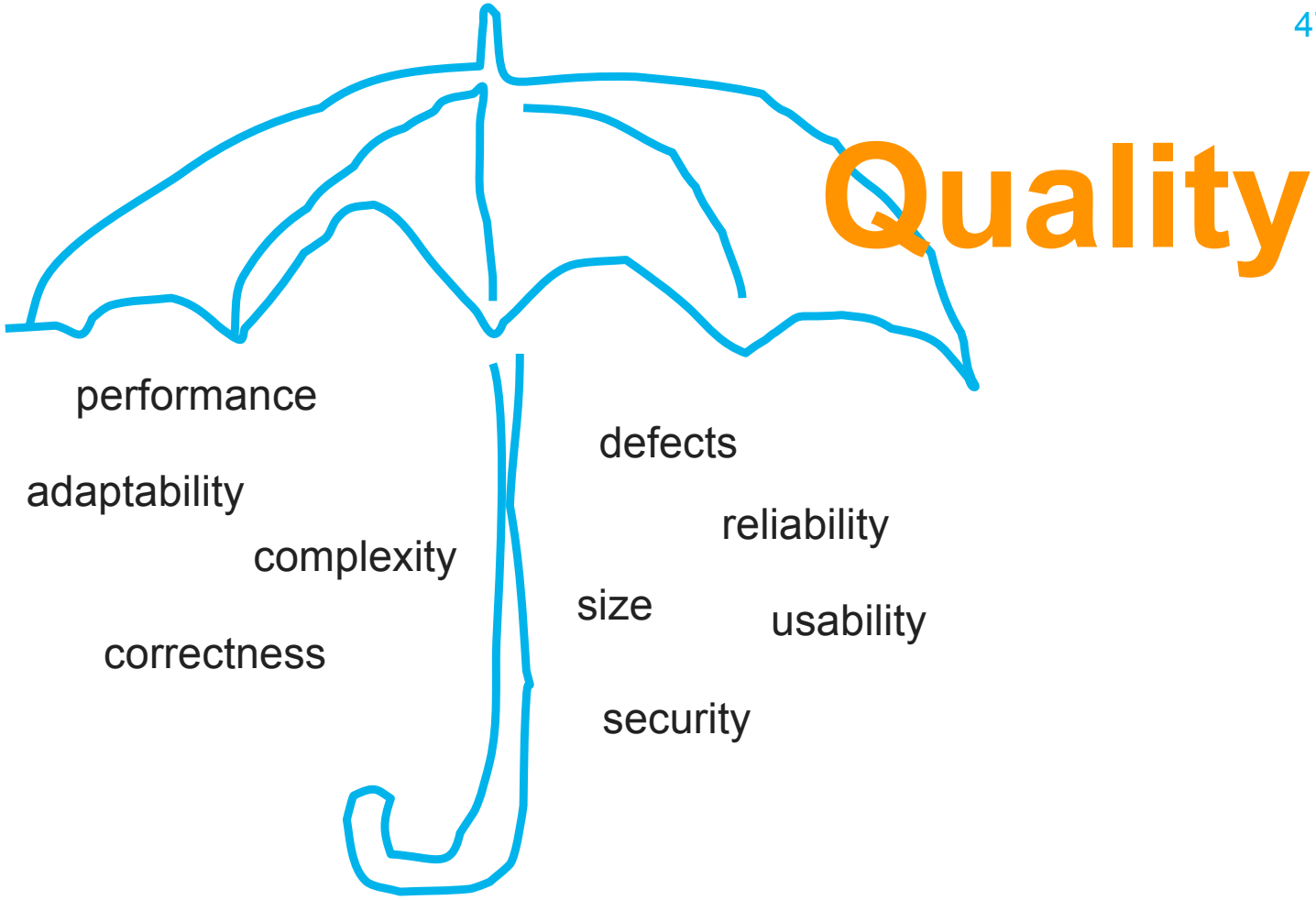
Structure of the lecture



METRICS & QUALITY

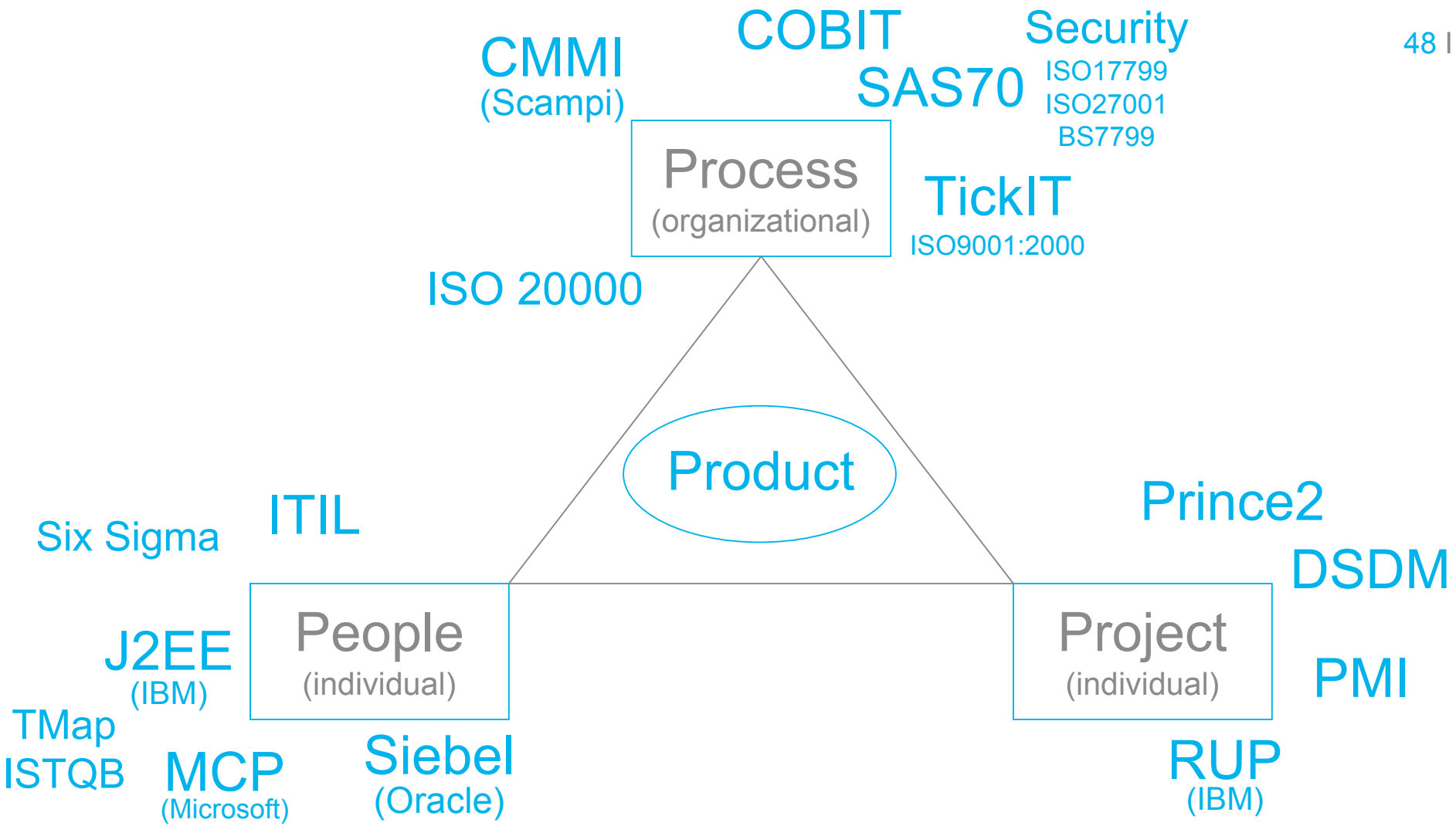
Software analysis
What?

Software Improvement Group



The bermuda triangle of software quality

Software Improvement Group



Software Improvement Group

Capability Maturity Model® Integration (CMMI®)

- "… is a process improvement approach that provides organizations with the essential elements of effective processes.." (SEI)
- CMMI for Development (CMMI-DEV), Version 1.2, August 2006.
- consists of 22 process areas with capability or maturity levels.
- CMMI was created and is maintained by a team consisting of members from industry, government, and the Software Engineering Institute (SEI)
- http://www.sei.cmu.edu/cmmi

The Standard CMMI Appraisal Method for Process Improvement (SCAMPI)

- "… is the official SEI method to provide benchmark-quality ratings relative to CMMI models."

Software Quality Process

Software Improvement Group

Software Engineering Institute | Carnegie Mellon

http://sas.sei.cmu.edu/pars/

Organization

Organization Name:	Accenture
Appraisal Sponsor Name:	Jack Ramsay, Marco Spaziani Testa, Maria Angeles Ramirez
Lead Appraiser Name:	John Voss
SEI Partner Name:	Accenture LLP

Model Scope and Appraisal Ratings

Level 2		Level 3		Level 4		Level 5	
Satisfied	REQM	Satisfied	RD	Out of Scope	OPP	Out of Scope	OID
Satisfied	PP	Satisfied	TS	Out of Scope	QPM	Out of Scope	CAR
Satisfied	PMC	Satisfied	PI				
Not Applicable	SAM	Satisfied	VER				
Satisfied	MA	Satisfied	VAL				
Satisfied	PPQA	Satisfied	OPF				
Satisfied	CM	Satisfied	OPD				
		Satisfied	OT				
		Satisfied	IPM				
		Satisfied	RSKM				
		Satisfied	DAR				

Organizational Unit Maturity Level Rating: 3
Additional Information for Appraisals Resulting in Capability or Maturity Level 4 or 5 Ratings:

Software Quality Process

Levels

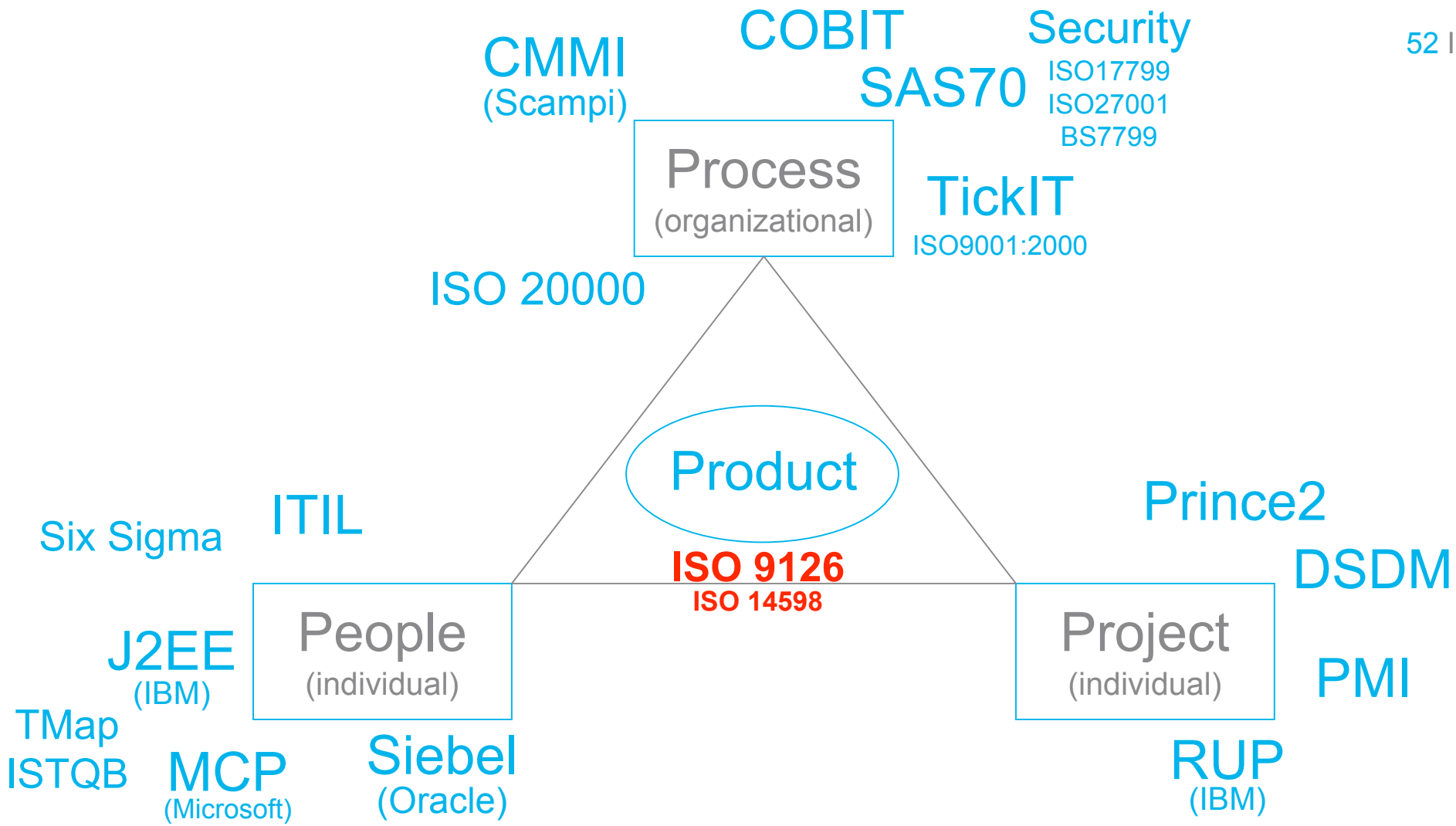
- L1: Initial
- L2: Managed
- L3: Defined
- L4: Quantitatively Managed
- L5: Optimizing

http://www.cmmi.de
(browser)

Process Areas

- Causal Analysis and Resolution
- Configuration Management
- Decision Analysis and Resolution
- Integrated Project Management
- Measurement and Analysis
- Organizational Innovation and Deployment
- Organizational Process Definition
- Organizational Process Focus
- Organizational Process Performance
- Organizational Training
- Product Integration
- Project Monitoring and Control
- CMMI Project Planning
- Process and Product Quality Assurance
- Quantitative Project Management
- Requirements Development
- Requirements Management
- Risk Management
- Supplier Agreement Management
- Technical Solution
- Validation
- Verification

The bermuda triangle of software quality



What is software quality?

What are the technical and functional aspects of quality?

How can technical and functional quality be measured?

Software product quality standards

ISO/IEC 9126

Software engineering -- Product quality

1. Quality model
2. External metrics
3. Internal metrics
4. Quality in use metrics

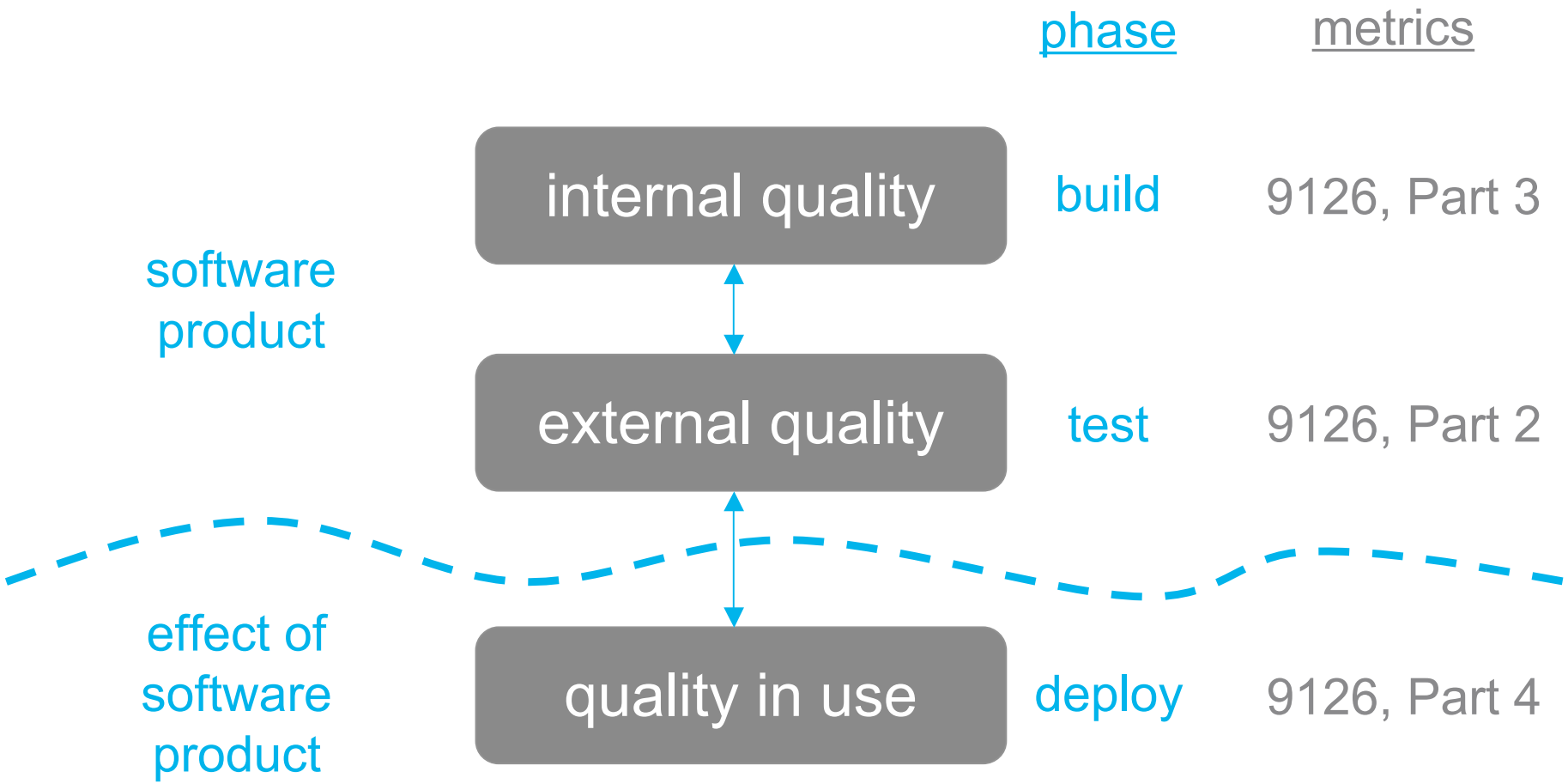
ISO/IEC 14598

Information technology -- Software product evaluation

1. General overview
2. Planning and management
3. Process for developers
4. Process for acquirers
5. Process for evaluators
6. Documentation of evaluation modules

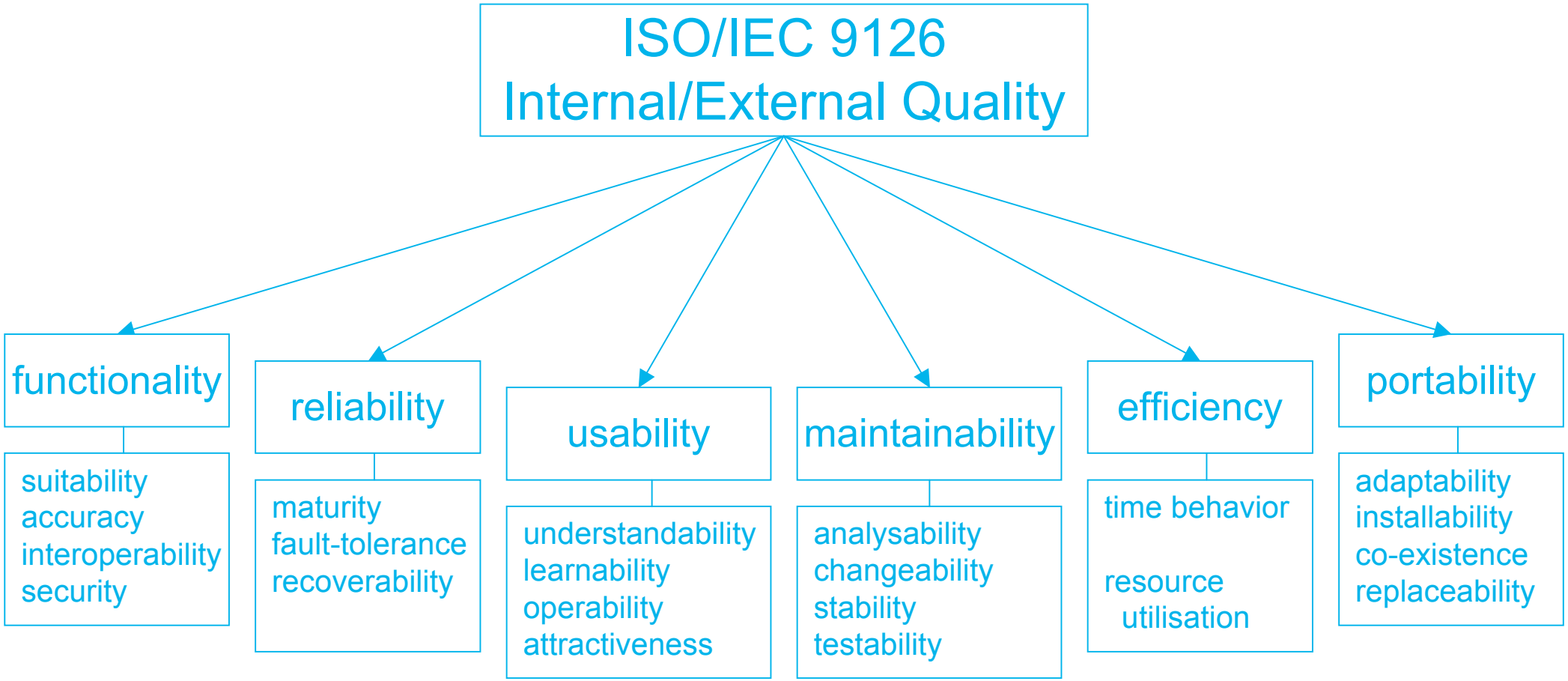
ISO/IEC 9126, Part 1
Quality perspectives

Software Improvement Group



ISO/IEC 9126, Part 1
Product quality model: internal and external

Software Improvement Group

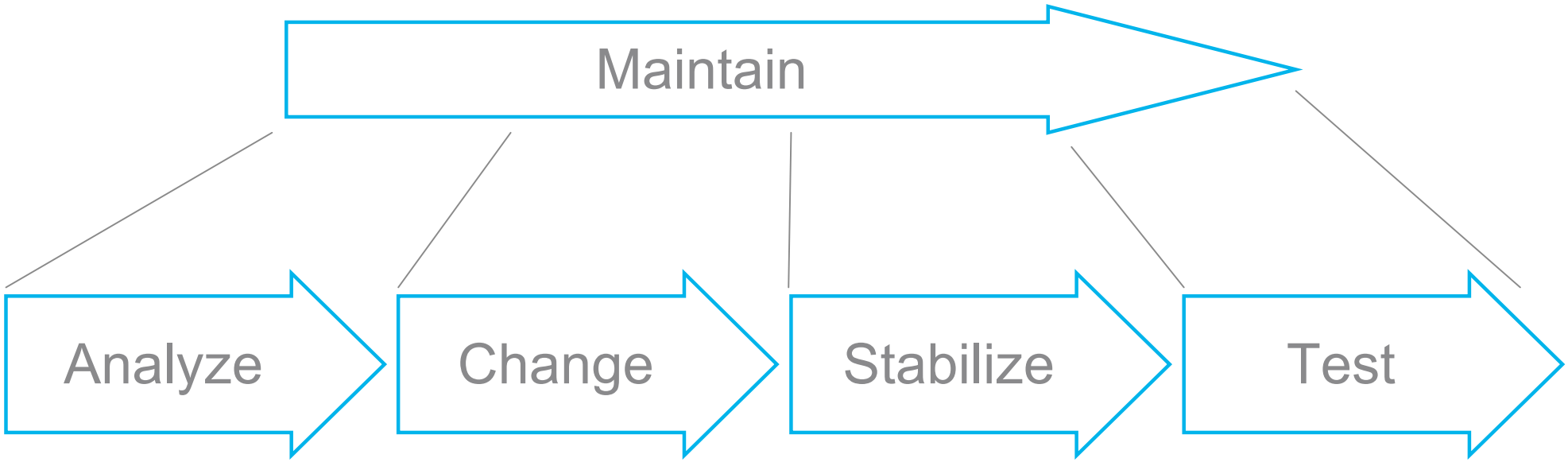


ISO 9126, Part 1
Maintainability (= evolvability)

Software Improvement Group

Maintainability =

- *Analyzability*: easy to understand where and how to modify?
- *Changeability*: easy to perform modification?
- *Stability*: easy to keep coherent when modifying?
- *Testability*: easy to test after modification?

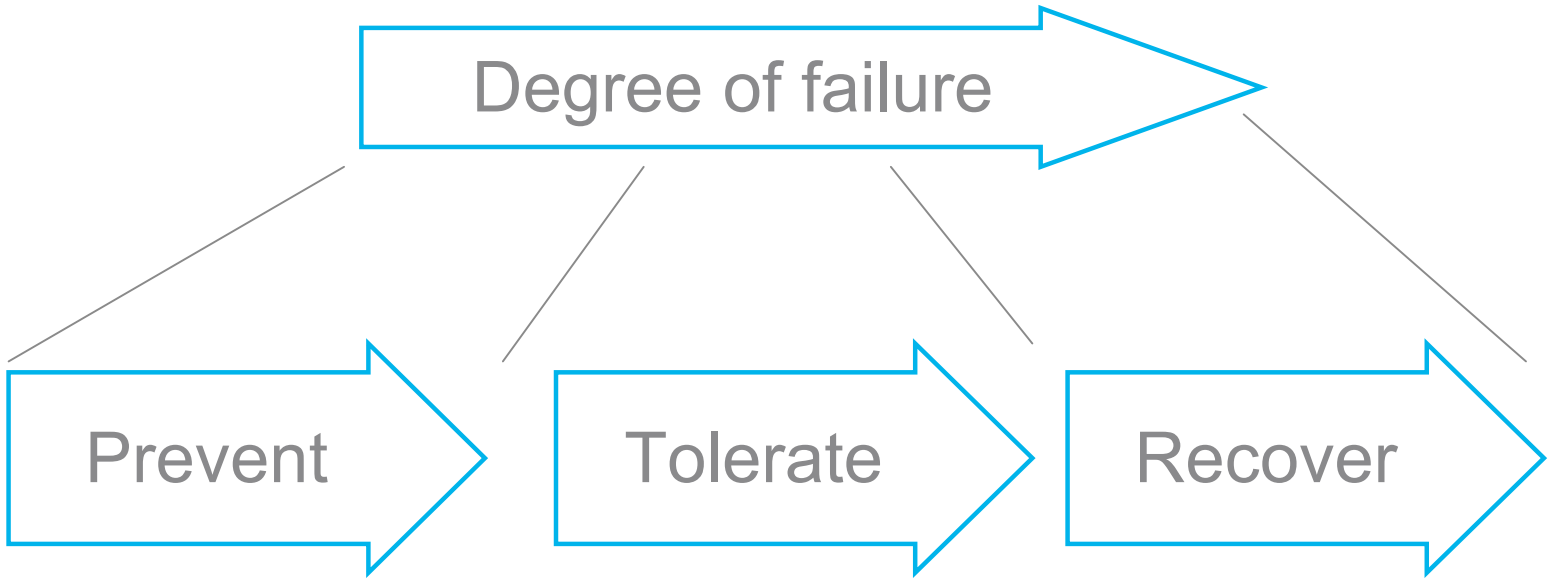


ISO 9126, Part 1
Reliability

Software Improvement Group

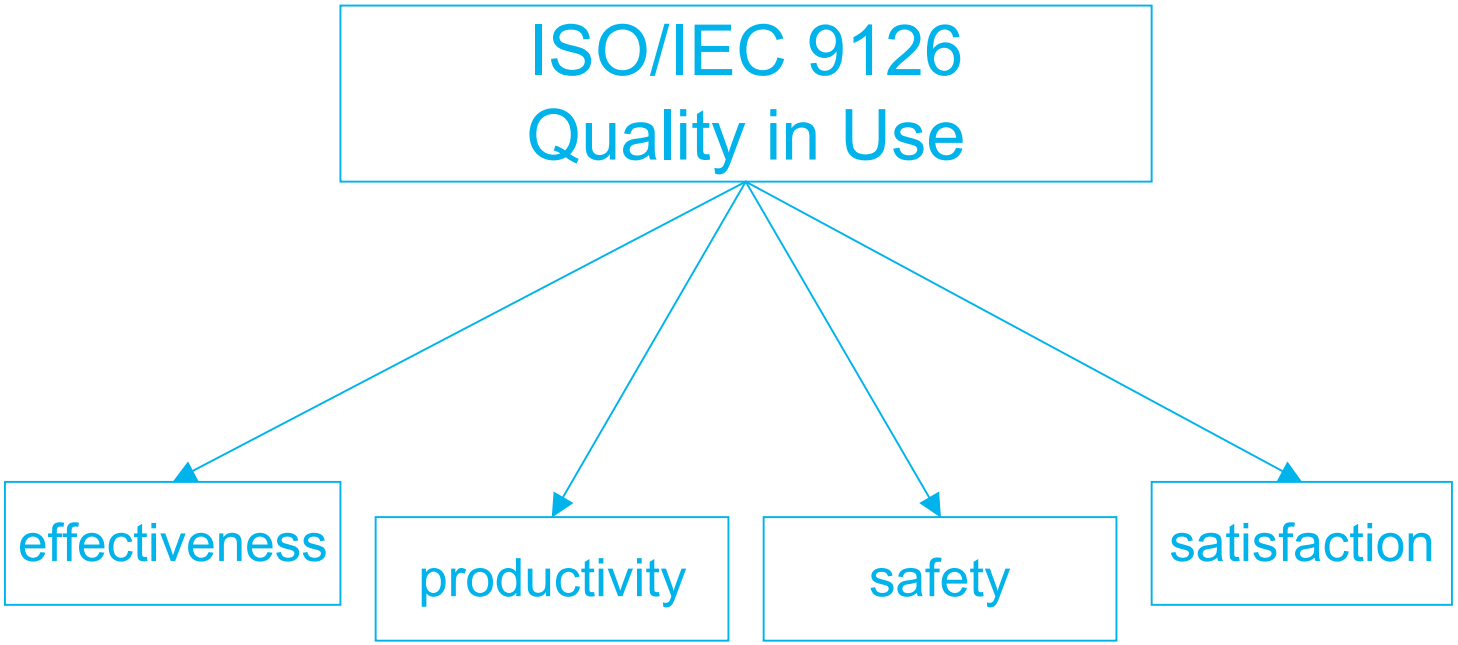
Reliability =

- *Maturity*: how much has been done to prevent failures?
- *Fault tolerance*: when failure occurs, is it fatal?
- *Recoverability*: when fatal failure occurs, how much effort to restart?



ISO/IEC 9126, Part 1
Product quality model: quality-in-use

Software Improvement Group



ISO 9126
Part 2,3: metrics

Software Improvement Group

External metrics, e.g.:

- Changeability: “change implementation elapse time”,
  time between diagnosis and correction
- Testability: “re-test efficiency”, time between correction and conclusion of test

Internal metrics, e.g.:

- Analysability: “activity recording”,
  ratio between actual and required number of logged data items
- Changeability: “change impact”,
  number of modifications and problems introduced by them

Critique

- Not pure *product* measures, rather *product in its environment*
- Measure *after* the fact
- No clear distinction between functional and technical quality

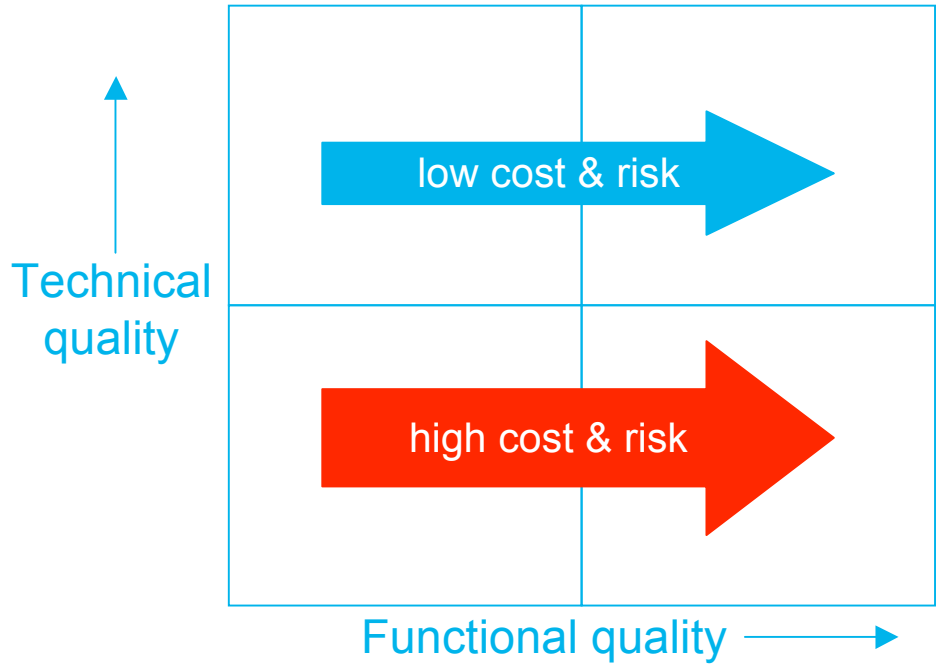
The issue

Software Improvement Group

- Companies innovate and change
- Software systems need to adapt in the same pace as the business changes
- Software systems that do not adapt lose their value
- The technical quality of software systems is a key element

Functional *vs* technical quality

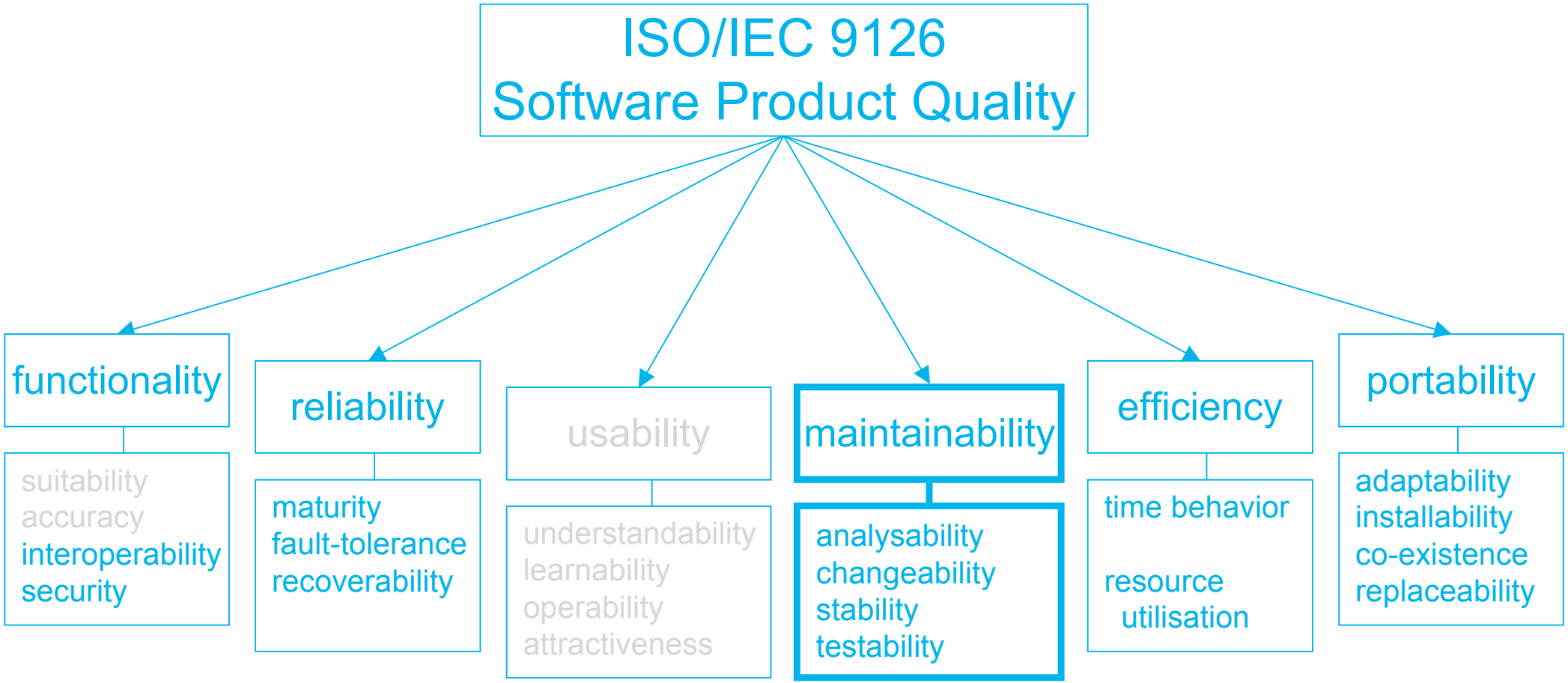
Software Improvement Group



Software with high technical quality can evolve with low cost and risk to keep meeting functional and non-functional requirements.

ISO/IEC 9126, Part 1
Product quality model: technical quality

Software Improvement Group



Software Improvement Group

What is software quality?

What are the functional and technical aspects of quality?

How can technical quality be measured?

A Challenge

Software Improvement Group

Use source code metrics to measure technical quality?

Plenty of metrics defined in literature

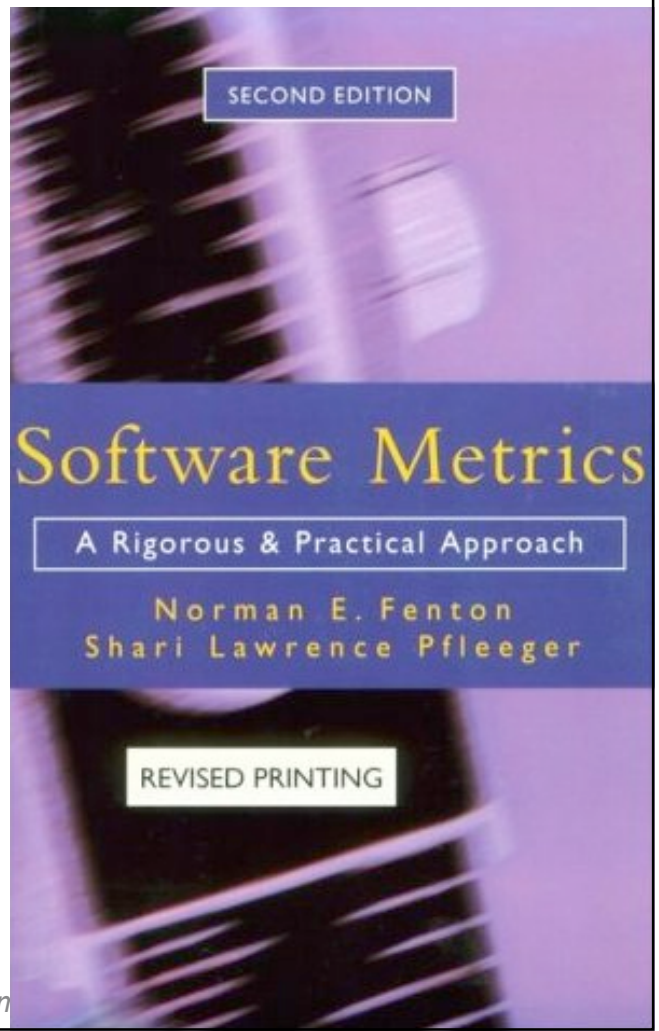
- LOC, cyclomatic complexity, fan in/out, coupling, cohesion, …
- Halstead, Chidamber-Kemener, Shepperd, …

Plenty of tools available

- Variations on Lint, PMD, FindBugs, …
- Coverity, FxCop, Fortify, QA-C, Understand, …
- Integrated into IDEs

But:

- Do they measure technical quality of a system?



Source code metrics
Lines of code (LOC)

- Easy! Or …

- SLOC = Source Lines of Code
  - Physical (≈ newlines)
  - Logical (≈ statements)
- Blank lines, comment lines, lines with only "}"
- Generated *versus* manually written

- Measure effort / productivity: specific to programming language

Source code metrics
Function Point Analysis (FPA)

Software Improvement Group

- A.J. Albrecht - IBM - 1979
- Objective measure of functional size

- Counted manually
  - IFPUG, Nesma, Cocomo
  - Large error margins

- Backfiring
  - Per language correlated with LOC
  - SPR, QSM

- Problematic, but popular for estimation

Table 2. Sample Function Point Calculations

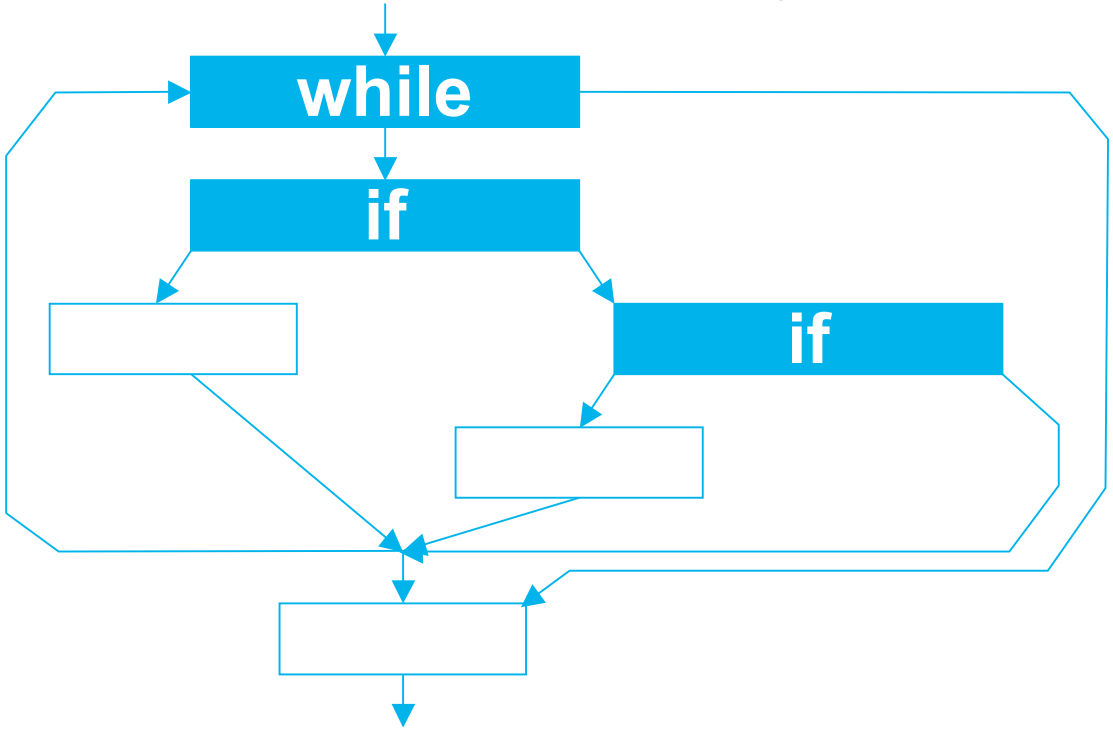
Raw Data	Weights		Function Points
1 Input	X 4	=	4
1 Output	X 5	=	5
1 Inquiry	X 4	=	4
1 Data File	X 10	=	10
1 Interface	X 7	=	7

Unadjusted Total			30
Compexity Adjustment			None
Adjusted Function Points			30

Source code metrics
Cyclomatic complexity

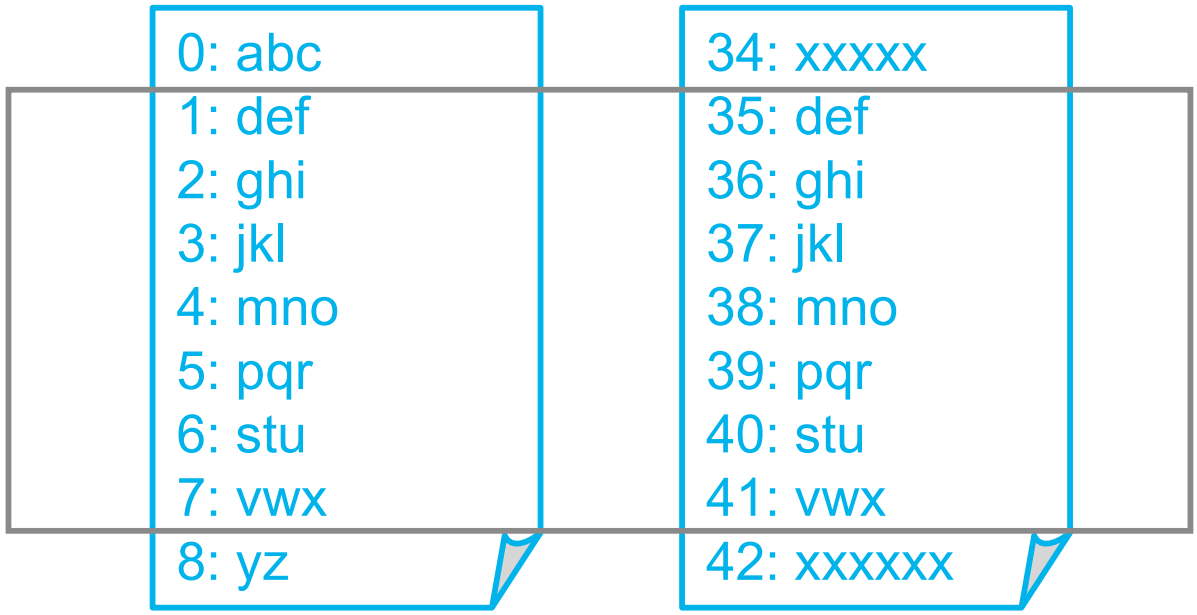
Software Improvement Group

- T. McCabe, *IEEE Trans. on Sw Engineering*, 1976
- Accepted in the software community
- Number of independent, non-circular paths per method
- Intuitive: number of decisions made in a method
- 1 + the number of if statements (and while, for, ...)



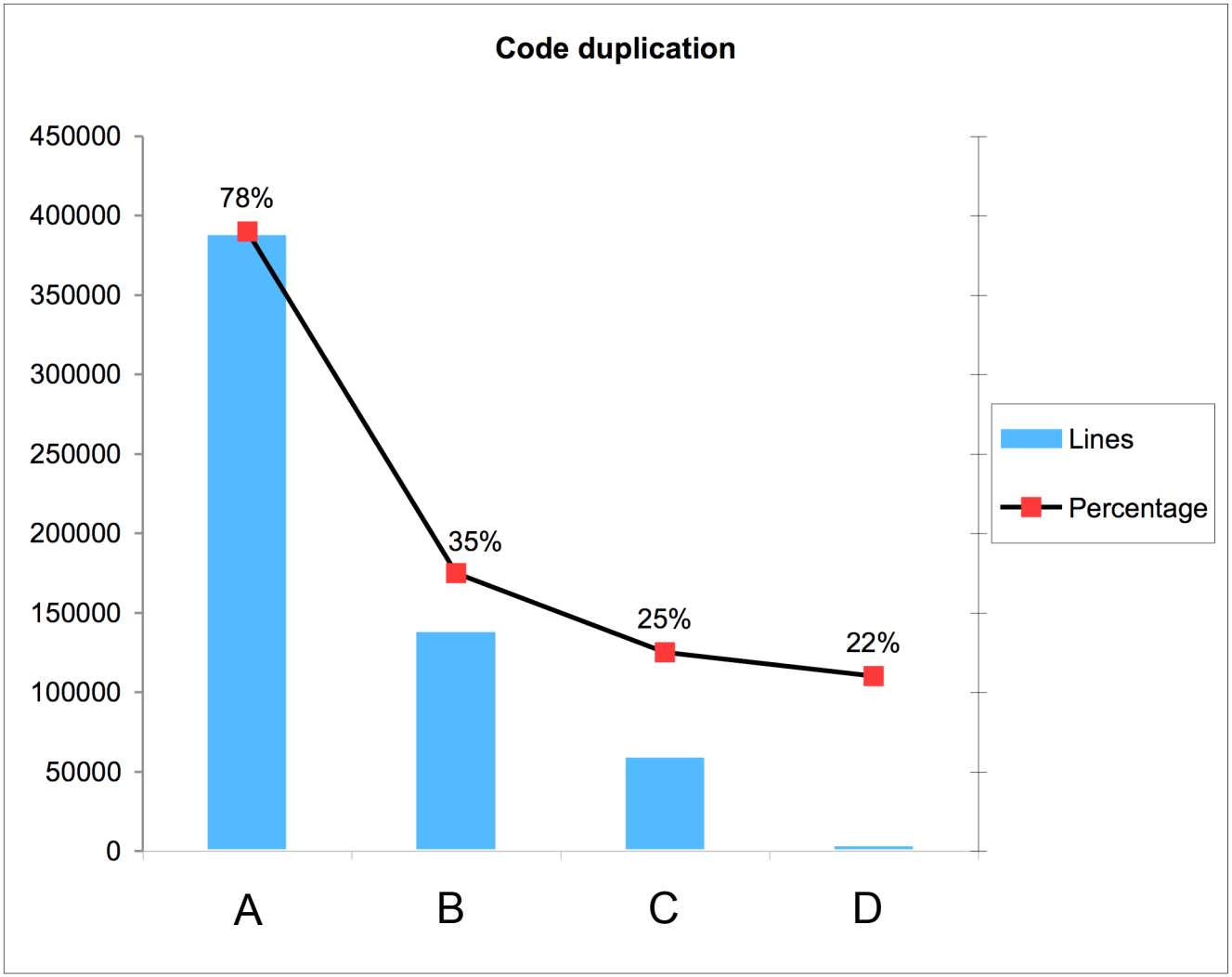
Software Improvement Group

Code duplication measurement



Number of duplicated lines: 14

Code duplication



Source code metrics
Coupling

Software Improvement Group

- Efferent Coupling (Ce)
  - How many classes do I depend on?
- Afferent Coupling (Ca)
  - How many classes depend on me?
- Instability = Ce/(Ca+Ce) ∈ [0,1]
  - Ratio of efferent *versus* total coupling
  - 0 = very stable = hard to change
  - 1 = very instable = easy to change

Figure 1. Coupling graph

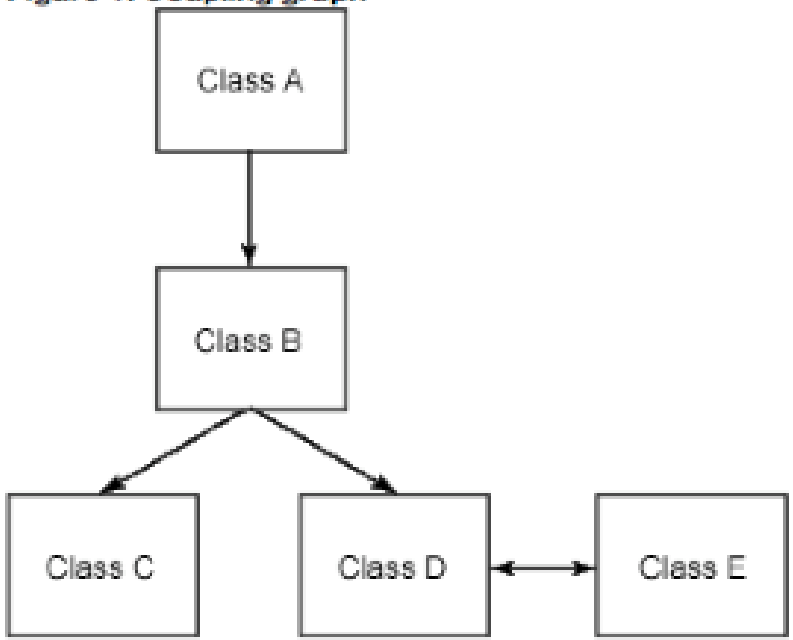
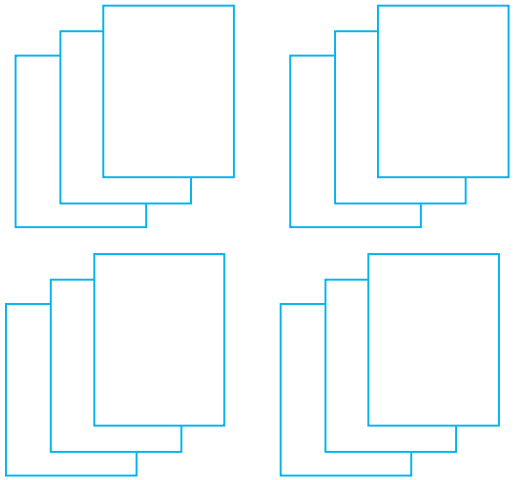


Table 1. Results of compiling a single class

Class to Compile	Other Classes Compiled	Afferent Couplings	Efferent Couplings	Instability Factor
A	B,C,D,E	0	4	1
B	C,D,E	1	3	0.75
C	-	2	0	0
D	E	3	1	0.25
E	D	3	1	0.25

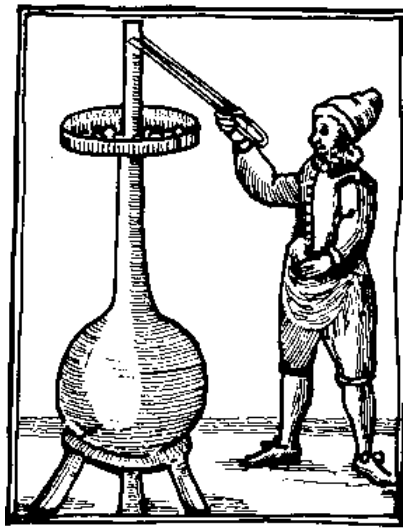
A Challenge
Do metrics measure technical quality?

Software Improvement Group



500.000 LOC Java code

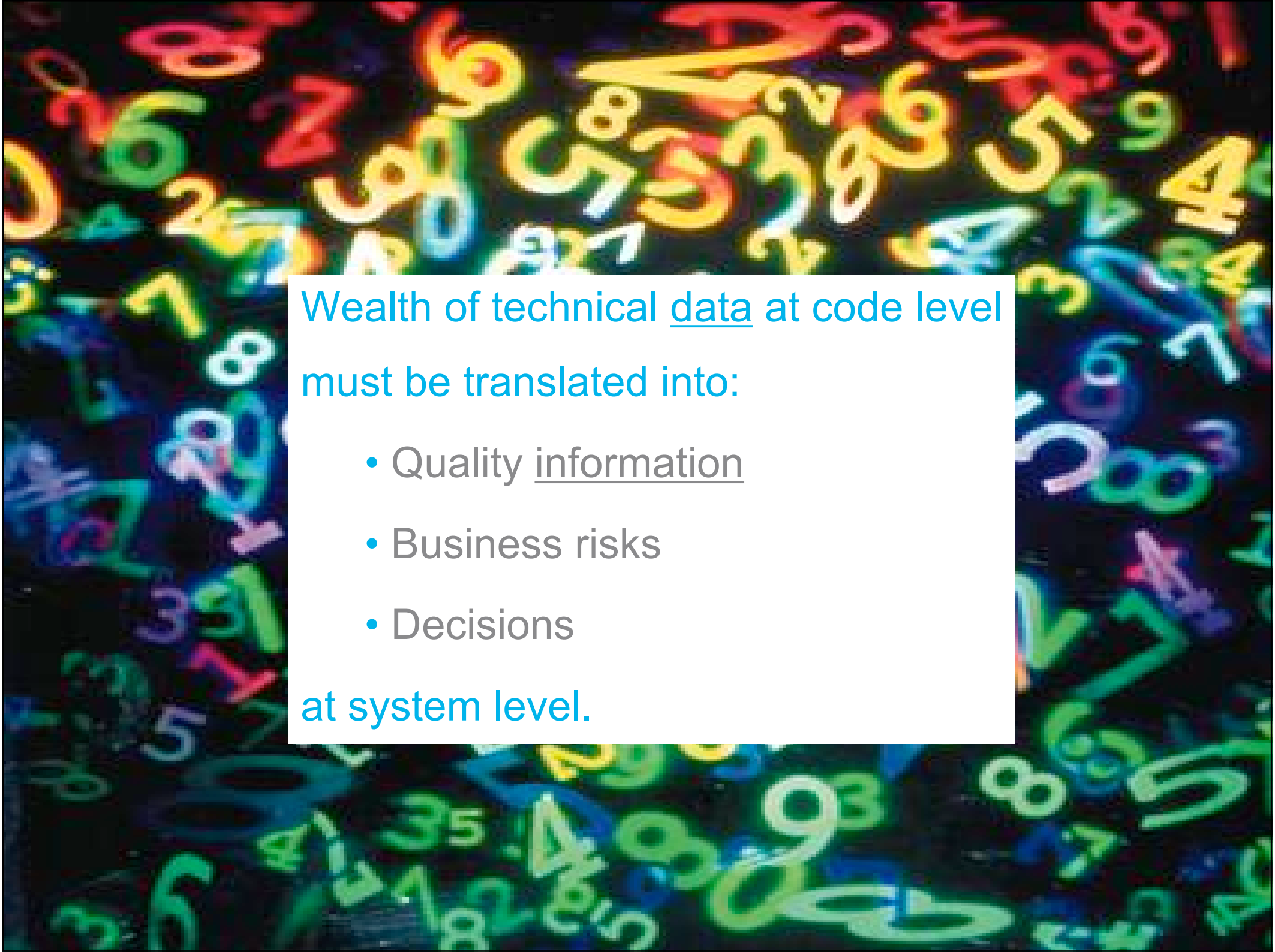
+



source code analyzer

=

?



Wealth of technical data at code level must be translated into:

- Quality information
- Business risks
- Decisions

at system level.

Source code metrics
Cyclomatic complexity

Software Improvement Group

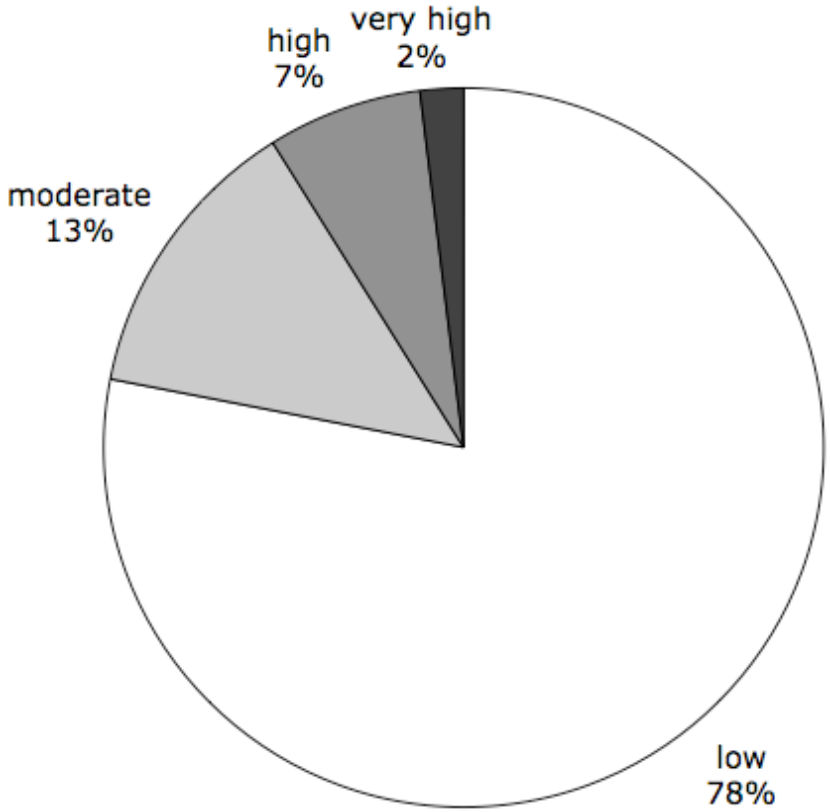
- T. McCabe, *IEEE Trans. on Sw Engineering*, 1976
- Accepted in the software community
- Academic: number of independent paths per method
- Intuitive: number of decisions made in a method
- Really, the number of if statements (and while, for, ...)
- Software Engineering Institute:

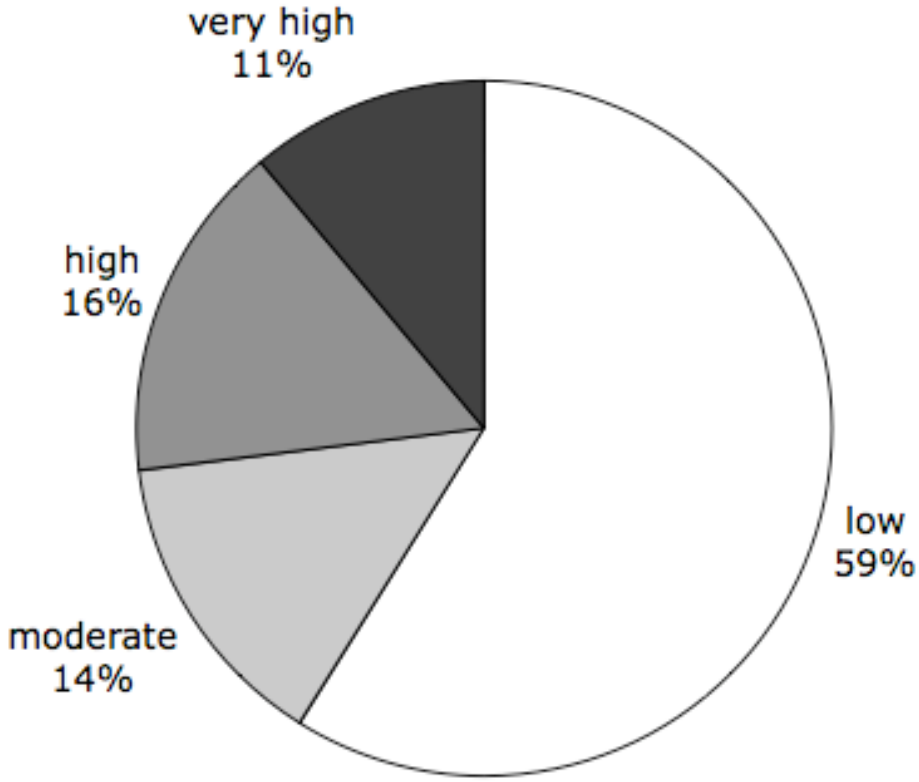
Table 4: Cyclomatic Complexity

Cyclomatic Complexity	Risk Evaluation
1-10	a simple program, without much risk
11-20	more complex, moderate risk
21-50	complex, high risk program
greater than 50	untestable program (very high risk)

Software Improvement Group

Aggregation by averaging is fundamentally flawed





Quality profiles, in general

Software Improvement Group

Input

- type Input metric = Map x (metric,LOC)

Risk groups

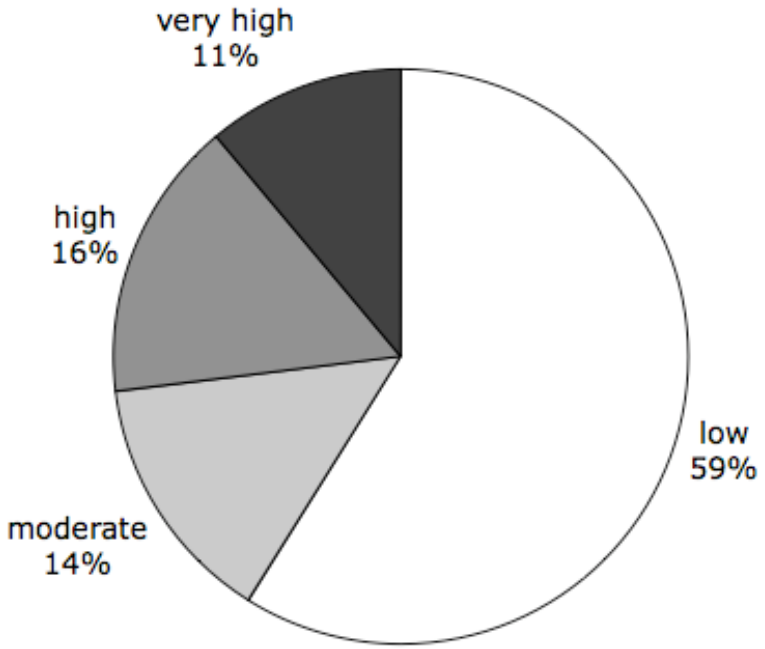
- type Risk = Low | Moderate | High | Very High
- risk :: metric → Risk

Output

- type ProfileAbs = Map Risk LOC
- type Profile = Map Risk Percentage

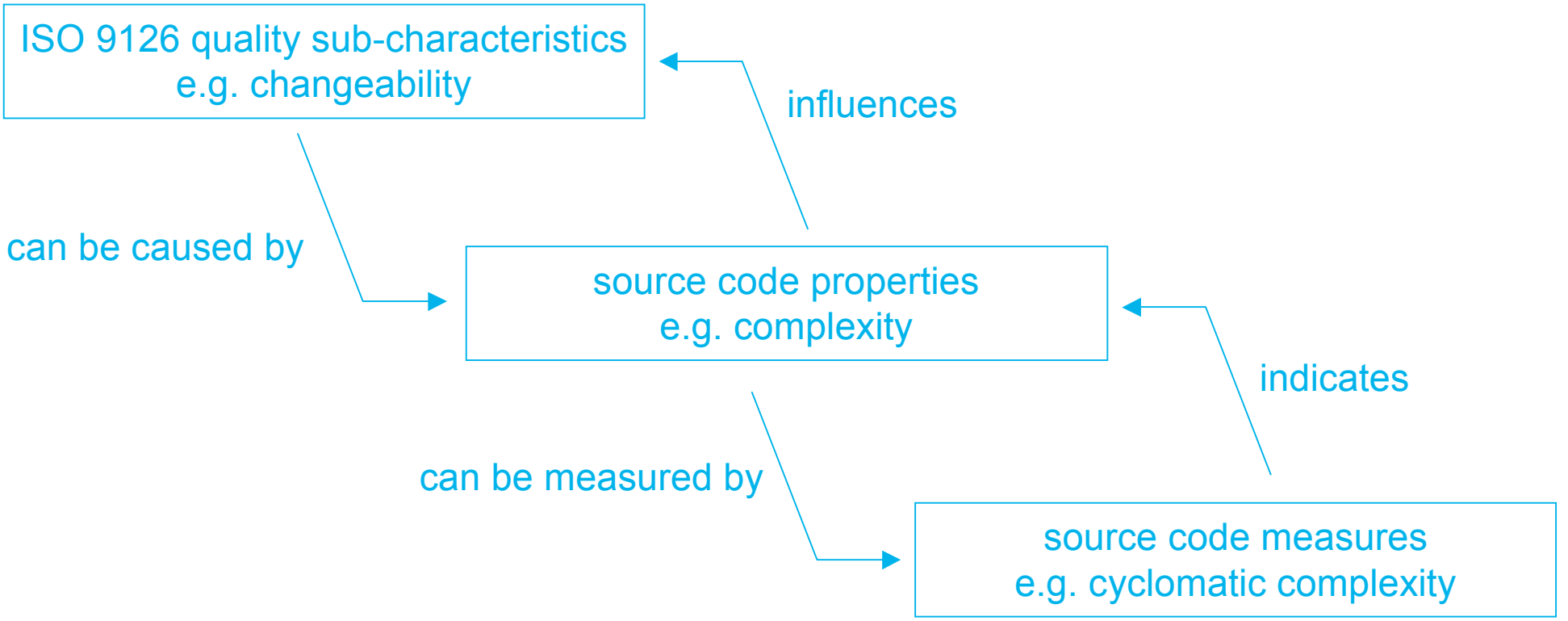
Aggregation

- profile :: Input metric → Profile



Combining metrics
The SIG approach

Software Improvement Group



Mapping source code properties onto quality sub-characteristics

Software Improvement Group

	Volume	Complexity	Unit size	Duplication	Unit testing	
Analysability	X		X	X		
Changeability		X		X		
Stability					X	
Testability		X	X		X	

Source code properties and metrics

Volume

- LOC, within the context of a single language
- Man years via backfiring function points

Complexity per unit

- McCabe's cyclomatic complexity, SEI risk categories, %LOC for each category

Duplication

- Duplicated blocks, threshold 6 lines, %LOC

Unit size

- LOC, risk categories, %LOC for each category

Unit testing

- Unit test coverage
- Number of assert statements (as validation)

Duplication

Software Improvement Group

	Volume	Complexity	Unit size	Duplication ★★★☆☆	Unit testing
Analysability	X		X	X	
Changeability		X		X	
Stability					X
Testability		X	X		X

Duplicate blocks

- Over 6 lines
- String comparison
- Remove leading spaces

Rank	duplication
★★★★★	0-3%
★★★★☆	3-5%
★★★☆☆	5-10%
★★☆☆☆	10-20%
★☆☆☆☆	20-100%

Complexity

Software Improvement Group

	Volume	Complexity ★★★☆☆	Unit size	Duplication	Unit testing	
Analysability	X		X	X		
Changeability		X		X		
Stability					X	
Testability		X	X		X	

Software Engineering Institute

complexity	risk
1-10	low
11-20	medium
21-50	high
>50	very high

	Maximum relative LOC		
Rank	**moderate**	**high**	**very high**
★★★★★	**25%**	**0%**	**0%**
★★★★☆	**30%**	**5%**	**0%**
★★★☆☆	**40%**	**10%**	**0%**
★★☆☆☆	**50%**	**15%**	**5%**
★☆☆☆☆	**-**	**-**	**-**

Rating example

Software Improvement Group

	Volume	Complexity	Unit size	Duplication	Unit testing	
	★★★★★	★☆☆☆☆	★★☆☆☆	★★☆☆☆	★★★☆☆	
Analysability	X		X	X		★★★☆☆
Changeability		X		X		★★☆☆☆
Stability					X	★★★☆☆
Testability		X	X		X	★★☆☆☆

That's all?

Software Improvement Group

Practical

- Fast, repeatable, technology independent
- Sufficiently accurate for our purposes
- Explainable

Beyond core model …

- Only one instrument in Software Risk Assessments and Software Monitor
- Weighting schemes
- Dynamic analysis
- Quality of process, people, project

Further reading

See

I. Heitlager, T. Kuipers, J. Visser.
A pragmatic model for measuring maintainability.
QUATIC 2007.

What is software quality?

What are the technical aspects of quality?

How can technical quality be measured?

Case 1
Curbing Erosion

Software Improvement Group

System

- About 15 years old
- Automates primary business process
- Maintenance has passed through various organizations
- New feature requests at regular intervals

Questions

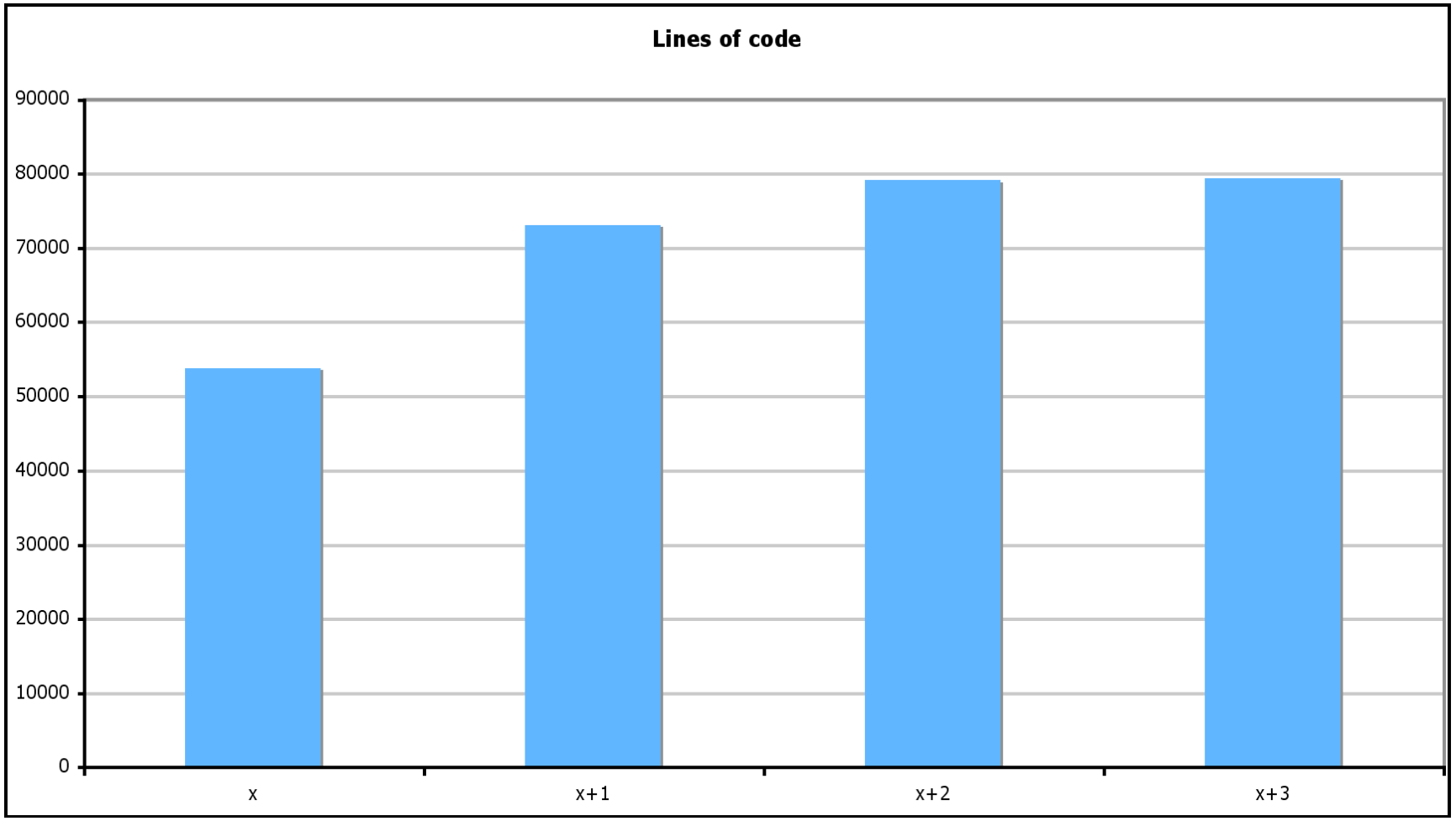
- Improve management's control over quality and associated costs

Metrics in this example

- Volume
- Duplication

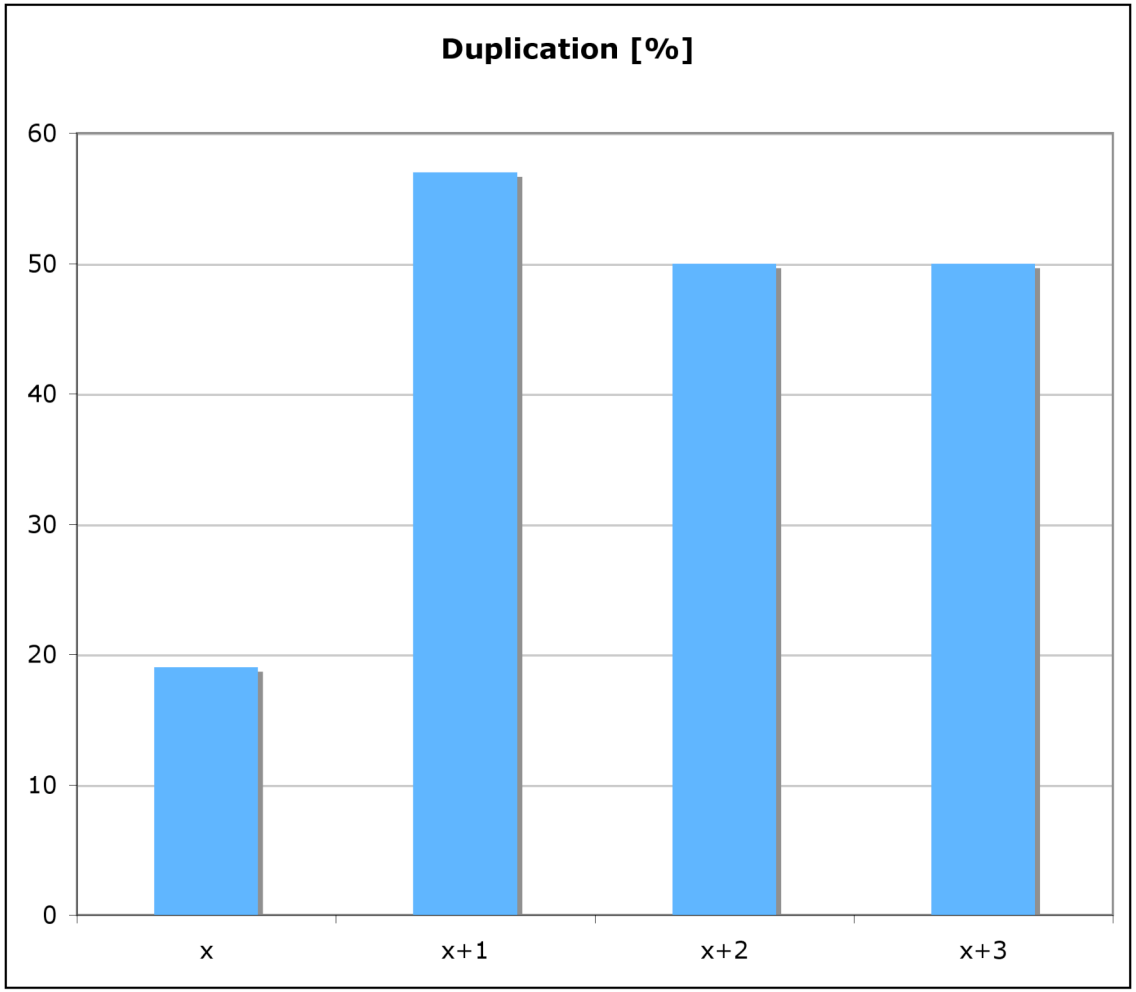
Case 1
Curbing Erosion

Software Improvement Group

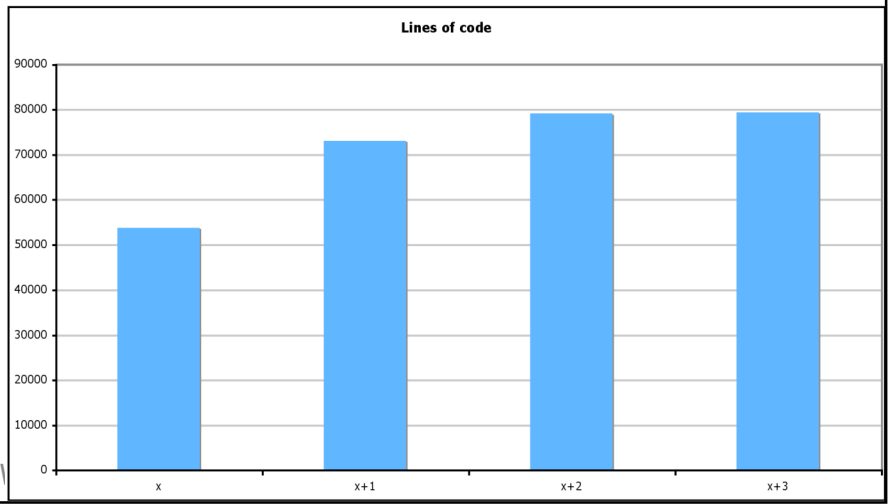


Case 1
Curbing erosion

Software Improvement Group



- All growth is caused by duplication
- There is no “real” productivity



Case 2
Systems accounting from code churn

Software Improvement Group

System

- 1.5 MLOC divided over 7000 files
- Estimated 240 people divided over 25 subcontractors

Questions

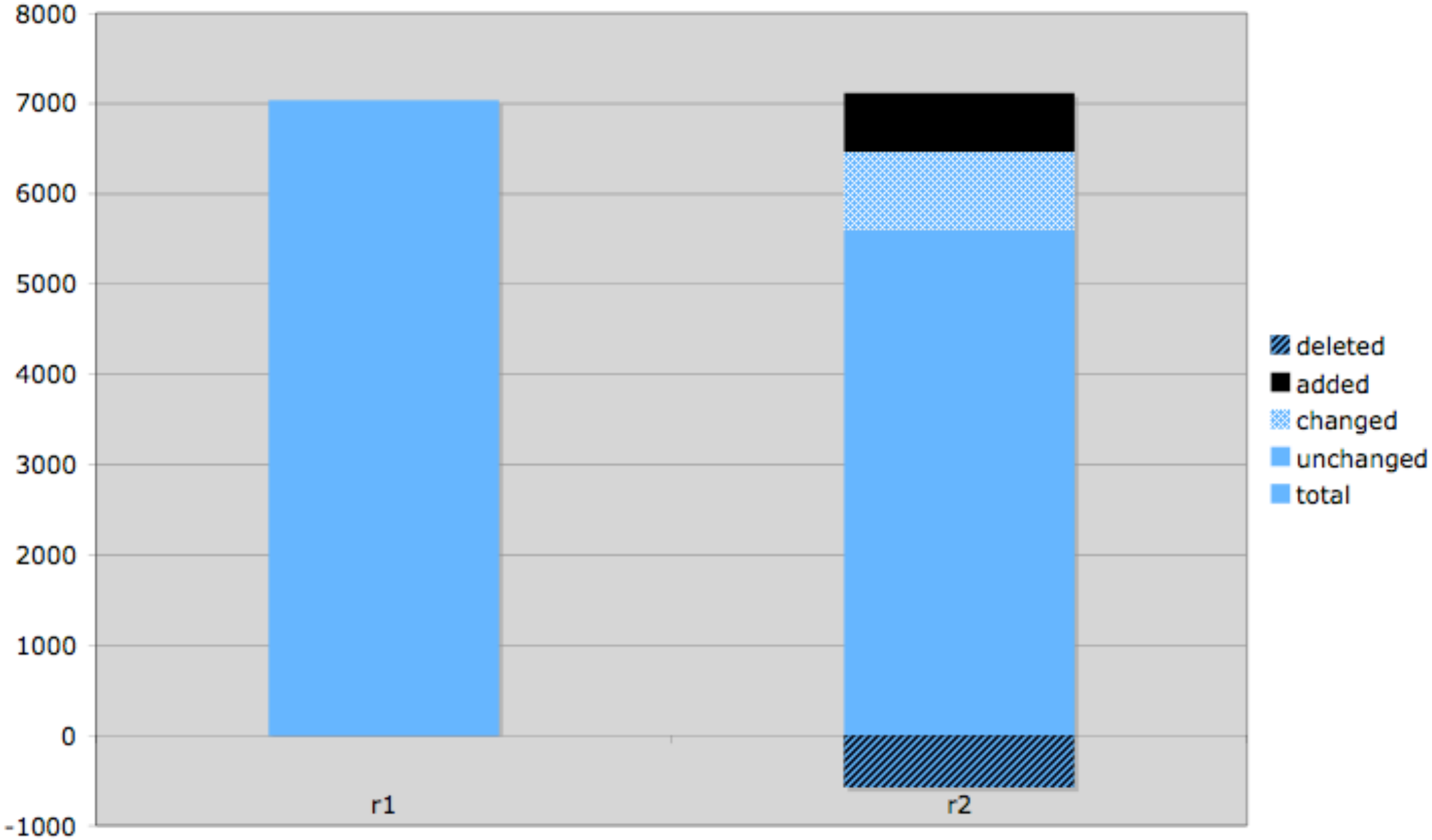
- Is staffing justified?

Metrics in this example

- Code churn = number of added, changed, deleted LOC

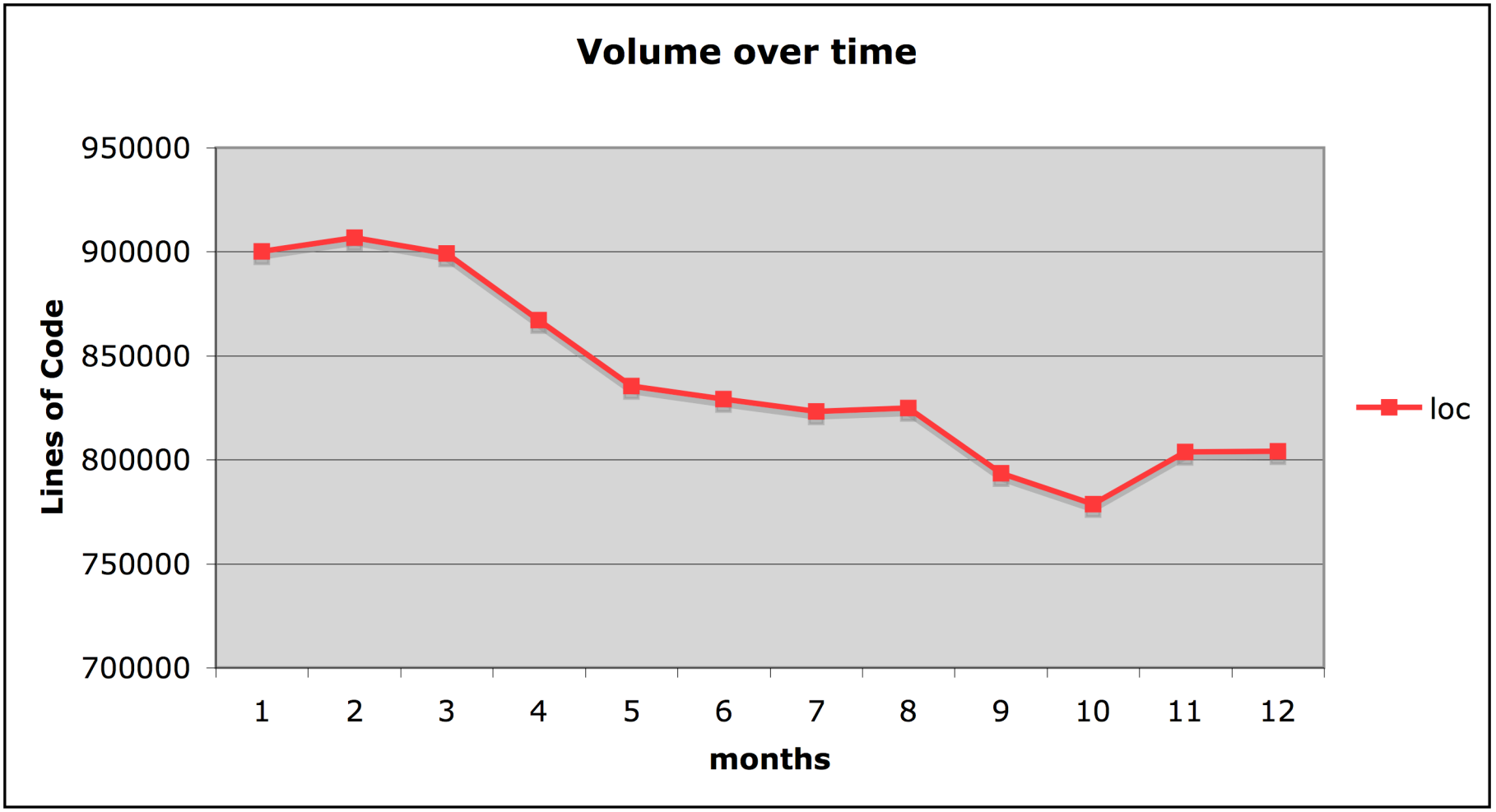
Case 2
Systems accounting from code churn

Software Improvement Group



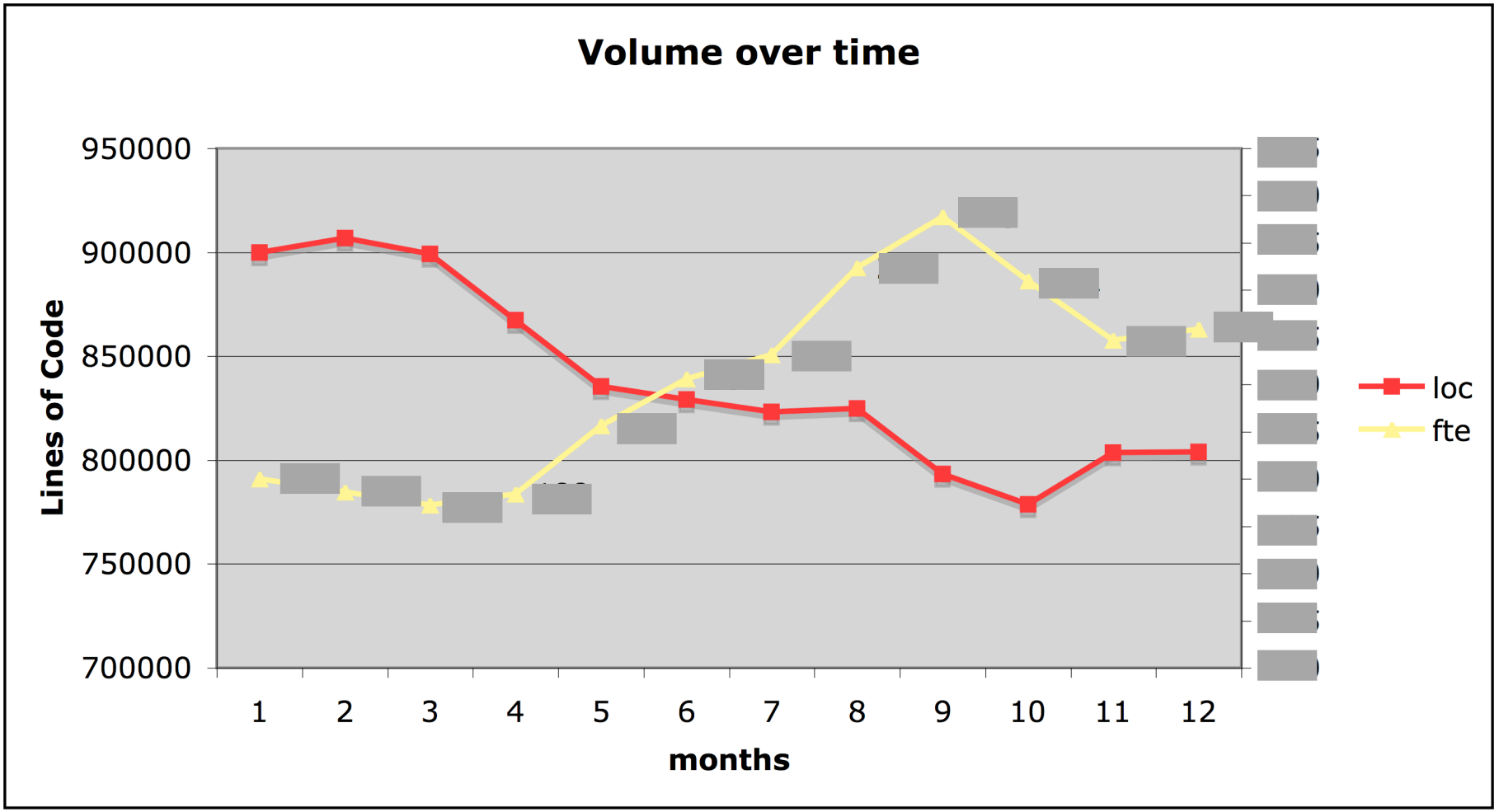
Case 2
Systems accounting from code churn

Software Improvement Group



Case 2
System accounting from code churn

Software Improvement Group



Case 3
Learn from failure

Software Improvement Group

System

- Electronic commerce
- Replacement for functionally identical system which failed in rollout
- Outsourced development

Questions

- Monitor productivity and quality delivered by the developer

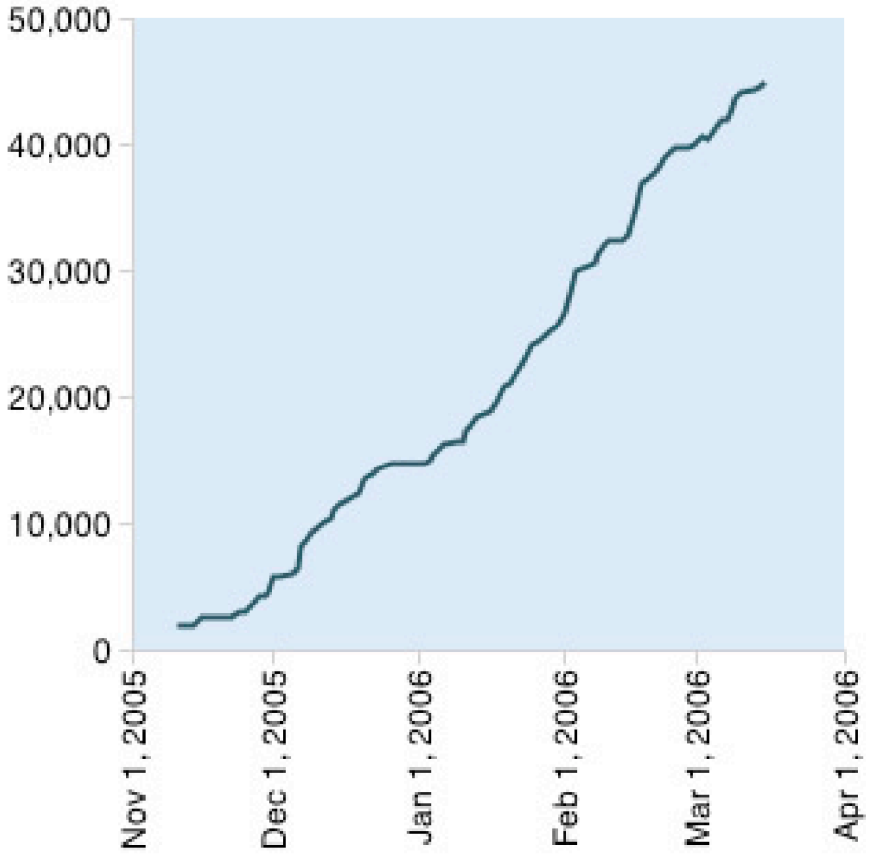
Metrics in this example

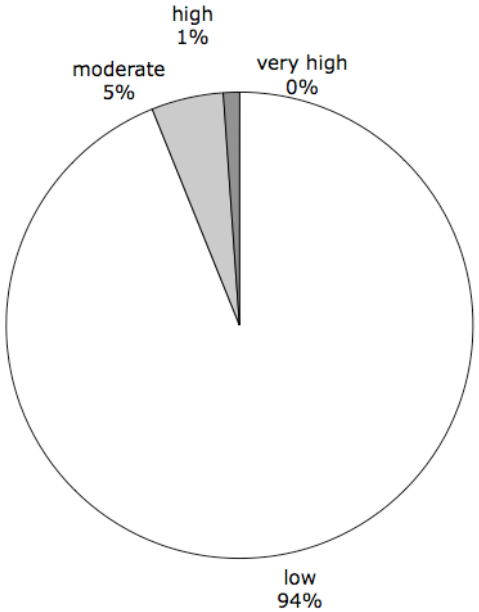
- Volume
- Complexity

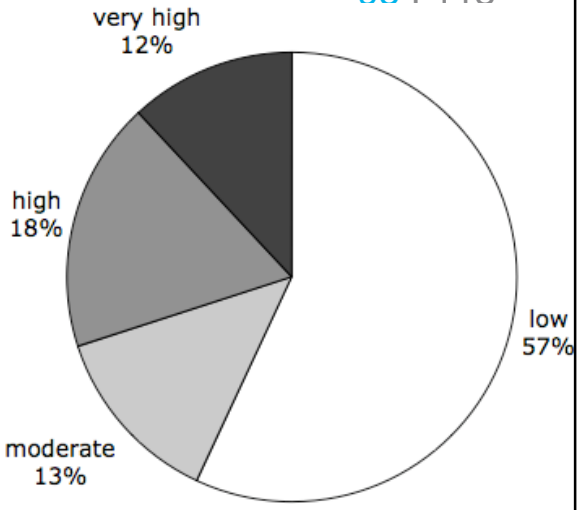
Case 3
Learn from failure

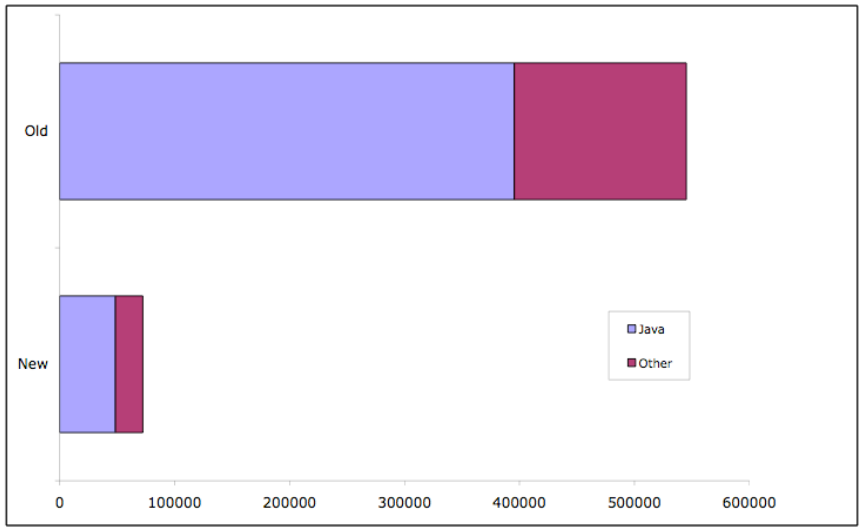
Software Improvement Group

Total lines sources









What should you remember (so far) from this lecture?

Testing

- Automated unit testing!

Patterns

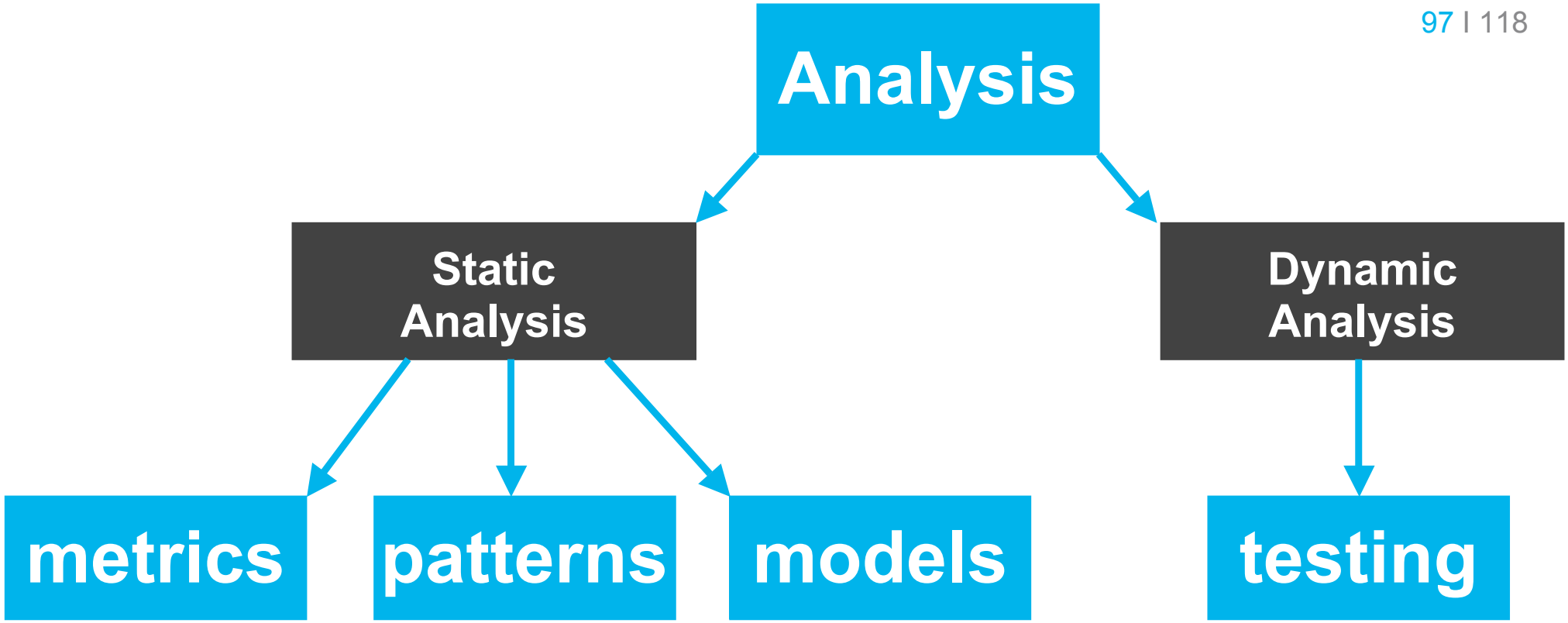
- Run tools!

Quality and metrics

- Technical quality matters in the long run
- A few simple metrics are sufficient
- If aggregated in well-chosen, meaningful ways
- The simultaneous use of distinct metrics allows zooming in on root causes

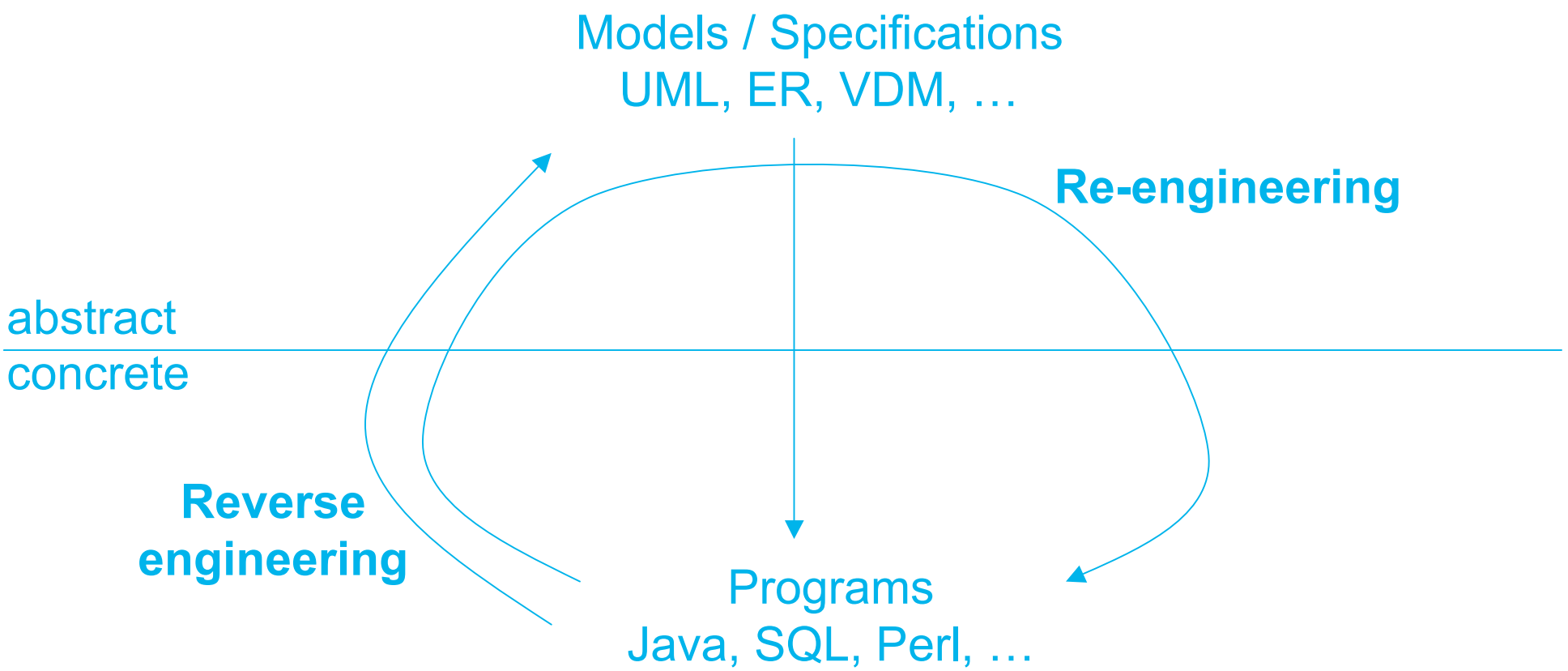
Structure of the lecture

Software Improvement Group



REVERSE ENGINEERING

Terminology



Dependencies and graphs

- Extraction, manipulation, presentation
- Graph metrics
- Slicing

Advanced

- Type reconstruction
- Concept analysis
- Programmatic join extraction

Reverse engineering trinity

Extraction

From program sources, extract basic information into an initial source model.

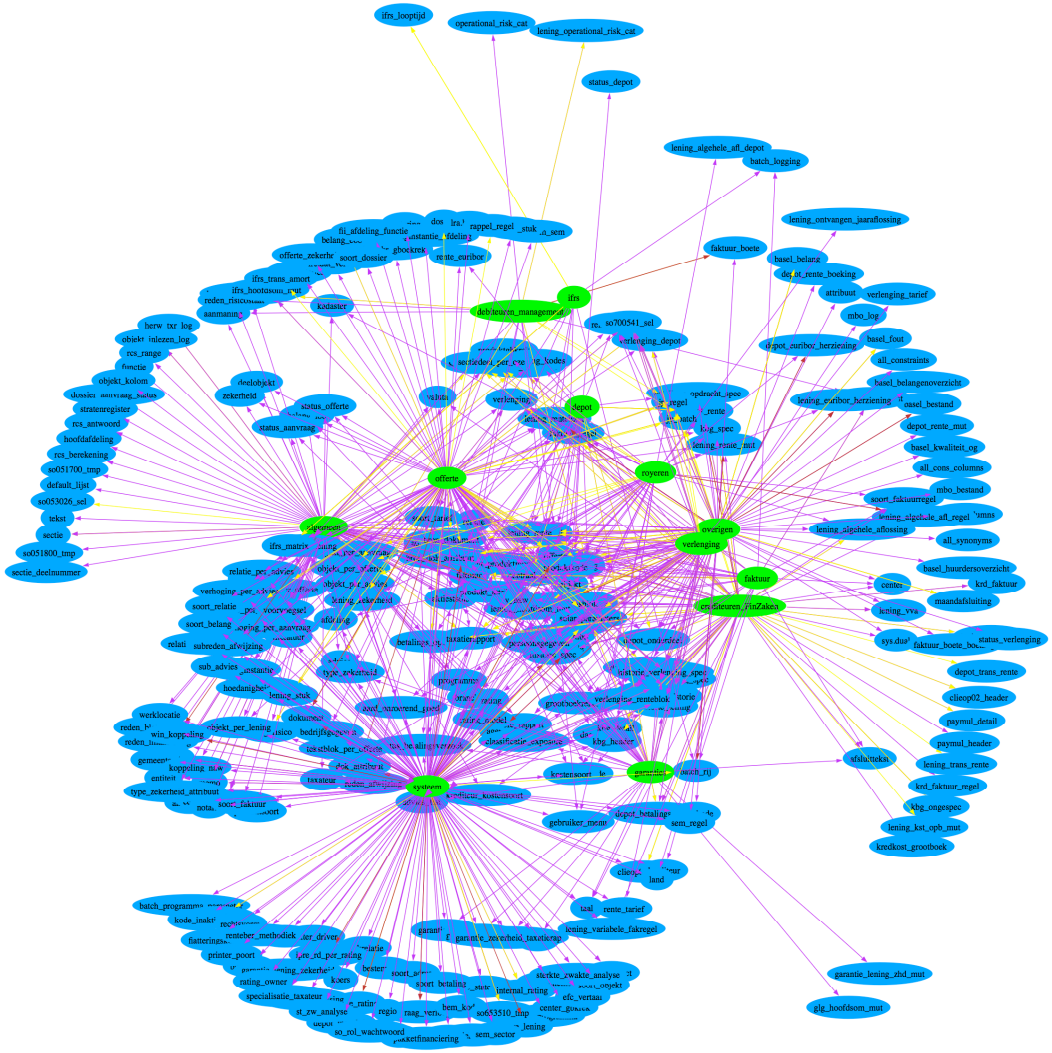
Manipulation

Combine, condense, aggregate, or otherwise process the basic information to obtain a derived source model.

Presentation

Visualize or otherwise present source models to a user.

Example



Green oval = module

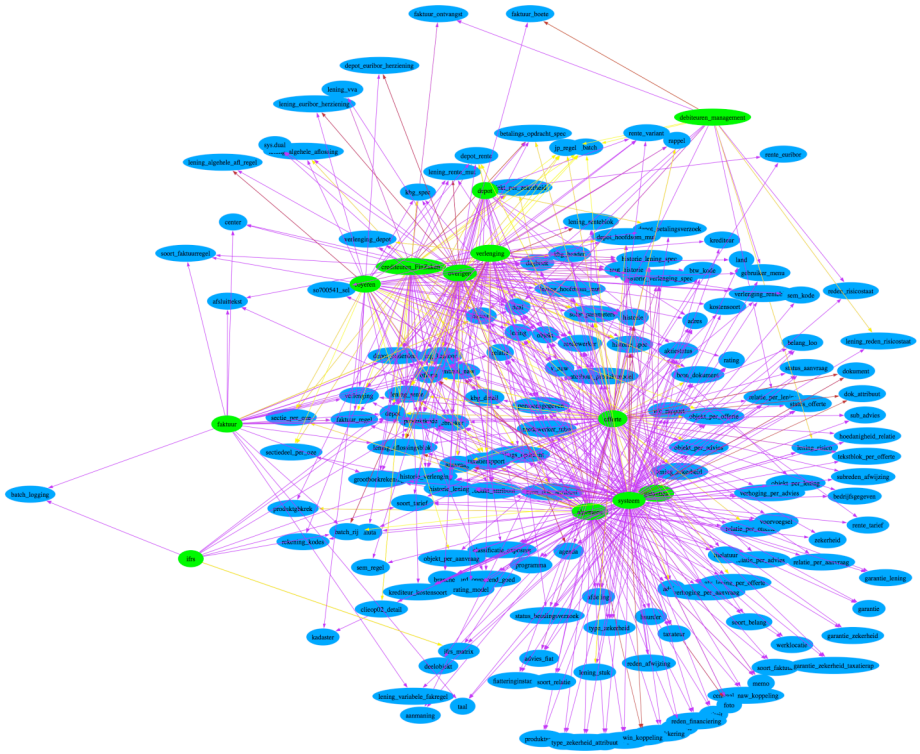
Blue oval = table

Purple arrow = select operation

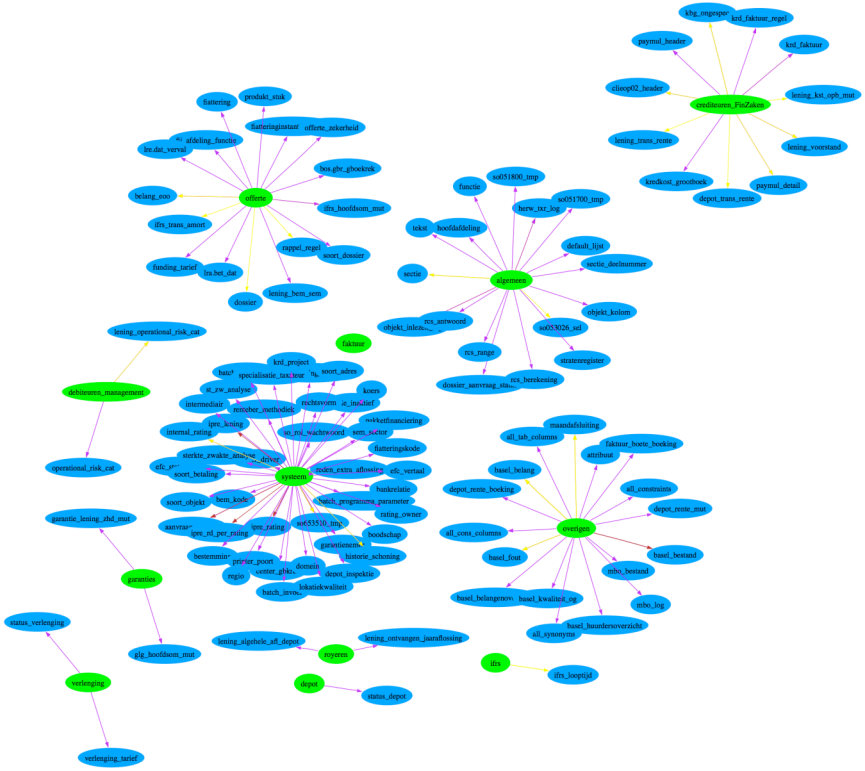
Yellow arrow = insert/update operation

Brown arrow = delete operation

Example



Tables used by multiple modules.



Tables used by a single module.

Relations and graphs

Relation

```
type Rel a b = Set (a,b)
```

set of pairs

Graph

```
type Gph a = Rel a a
```

endo-relation

Labeled relation

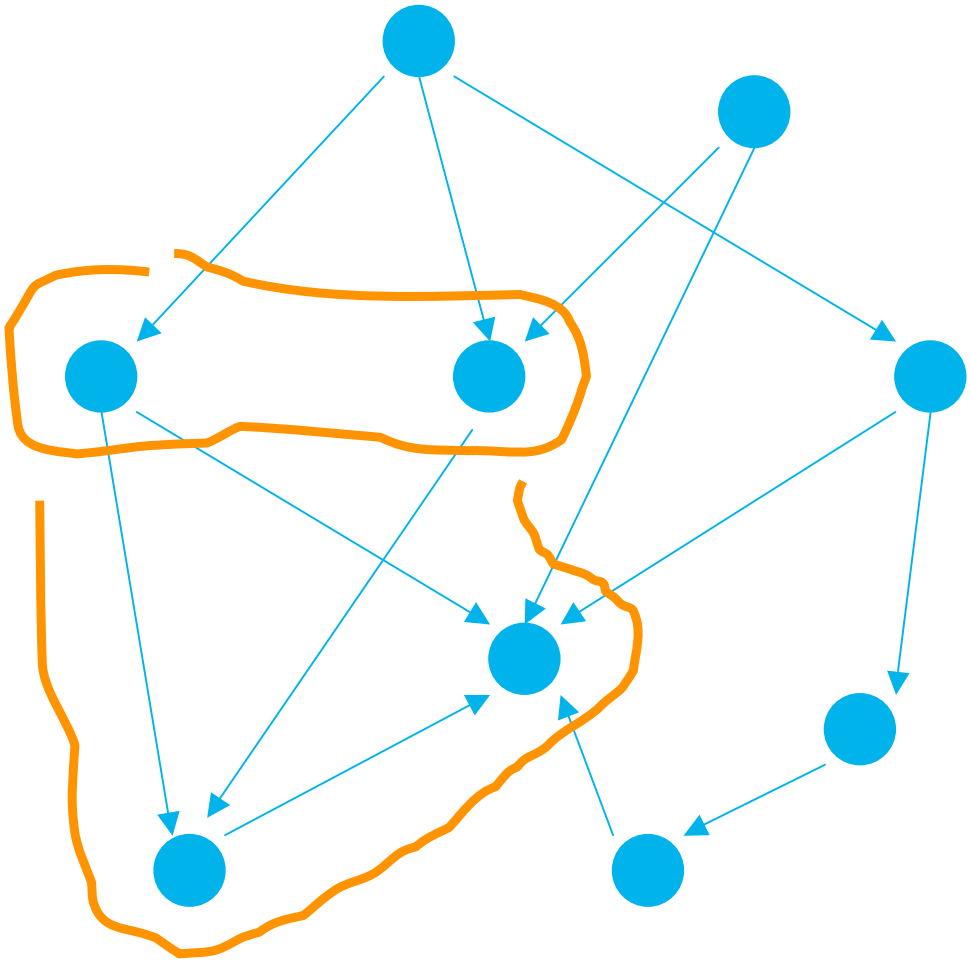
```
type LRel a b l = Map (a,b) l
```

map from pairs

Note

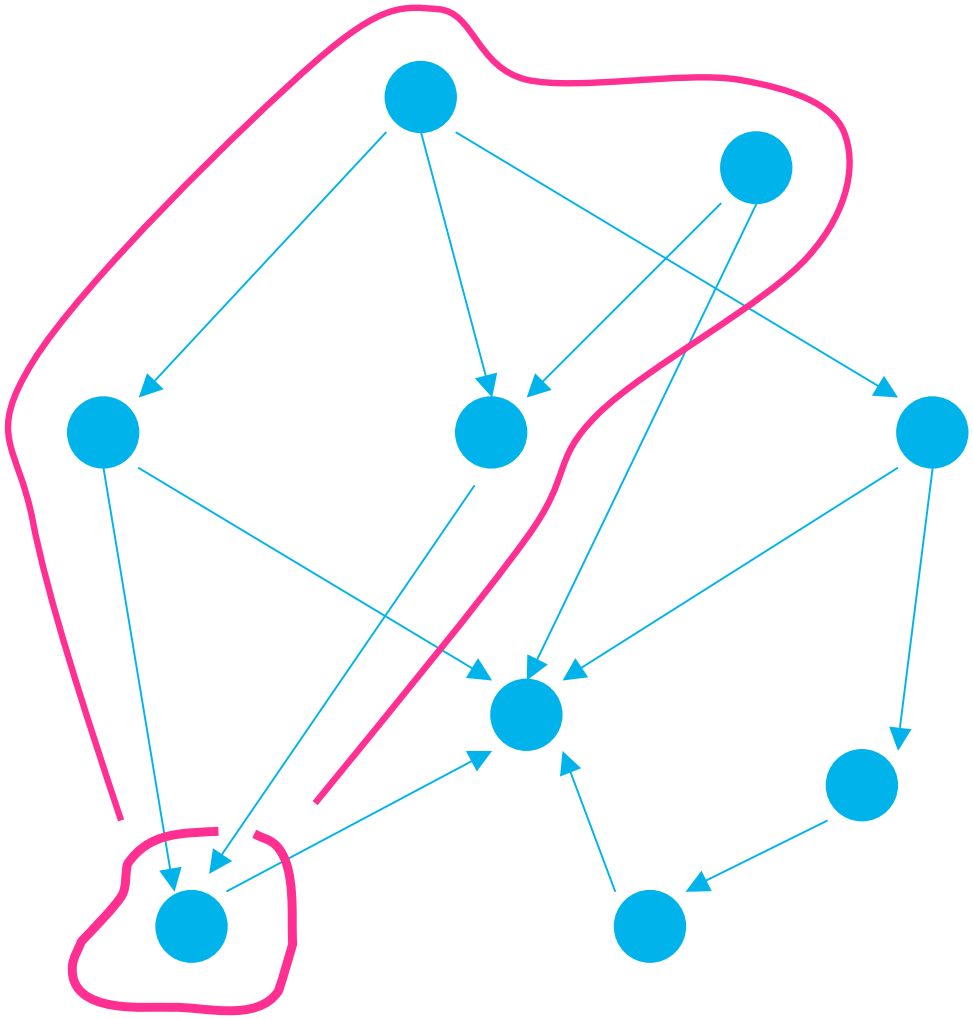
```
Rel a b = Set(a,b)= Map(a,b)()= LRel a b ()
```

Slicing (forward)



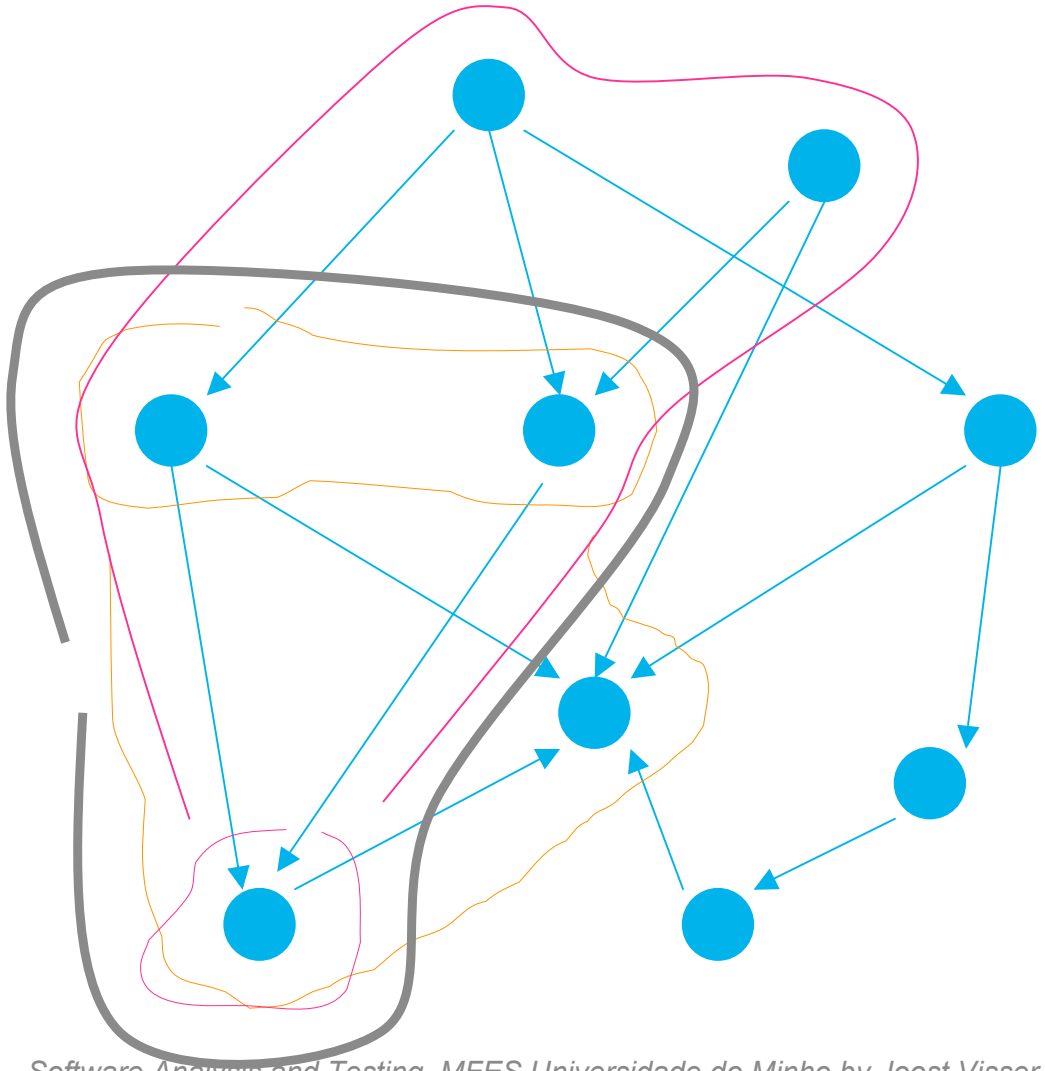
Slicing (backward)

Software Improvement Group

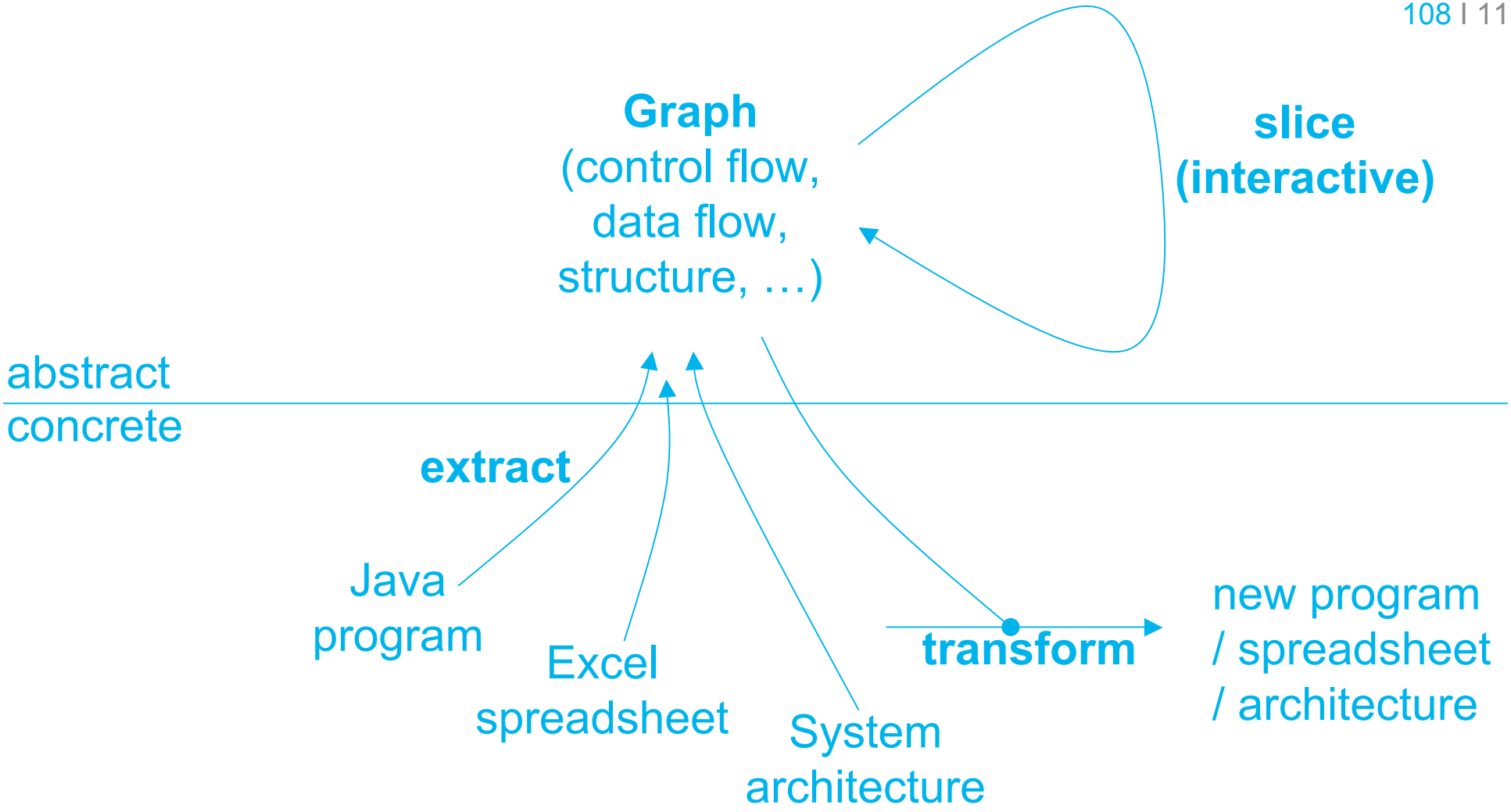


Chop
= Forward ∩ Backward

Software Improvement Group



Generic slicing



Further reading

See

Arun Lakhotia.
Graph theoretic foundations of program slicing and integration.
The Center for Advanced Computer Studies, University of Southwestern Louisiana.
Technical Report CACS TR-91-5-5, 1991.

Type reconstruction (from type-less legacy code)

Software Improvement Group

See

- Arie van Deursen and Leon Moonen. *An empirical Study Into Cobol Type Inferencing*. Science of Computer Programming **40**(2-3):189-211, July 2001

Basic idea

1. Extract basic relations (entities are variables)
   - assign: ex. `a := b`
   - expression: ex. `a <= b`
   - arrayIndex: ex. `A[i ]`
2. Compute derived relations
   - typeEquiv: variables belong to the same type
   - subtypeOf: variables belong to super/subtype
   - extensional notion of type: set of variables

Type reconstruction (from type-less legacy code)

Software Improvement Group

Pseudo code from paper

arrayIndexEquiv := arrayIndex^{-1} ∘ arrayIndex

typeEquiv := arrayIndexEquiv ∪ expression

subtypeOf := assign

repeat

  subtypeEquiv := equiv(subtypeOf+ ∩ (subtypeOf+)$^{-1}$)

  typeEquiv := equiv(typeEquiv ∪ subtypeEquiv)

  subtypeOf := subtypeOf \ typeEquiv

  subtypeOf := subtypeOf ∪ subtypeOf ∘ typeEquiv ∪ typeEquiv ∘ subtypeOf

until fixpoint of (typeEquiv, subtypeOf)

Type reconstruction (from type-less legacy code)

Software Improvement Group

Data

```
type VariableGraph v array
  = (Rel v v, Rel v array, Rel v v)

type TypeGraph x
  = (Rel x x, Rel x x)  -- subtypes and type equiv
```

Operation

```
typeInference
  :: (Ord v, Ord array) =>
     VariableGraph v array -> TypeGraph v
```

Formal concept analysis

See

- Christian Lindig. *Fast Concept Analysis*. In Gerhard Stumme, editors, *Working with Conceptual Structures - Contributions to ICCS 2000*, Shaker Verlag, Aachen, Germany, 2000.

Basic idea

1. Given formal context
   - matrix of objects vs. properties
2. Compute concept lattice
   - a concept = (extent,intent)
   - ordering is by sub/super set relation on intent/extent

Used in many fields, including program understanding.

Formal concept analysis pseudo-code (1/2)

Software Improvement Group

NEIGHBORS $((G, M), (\mathcal{G}, \mathcal{M}, \mathcal{I}))$

```
1   min ← 𝒢 \ G
2   neighbors ← ∅
3   foreach g ∈ 𝒢 \ G do
4      M1 ← (G ∪ {g})′
5      G1 ← M1′
6      if ((min ∩ (G1 \ G \ {g})) = ∅) then
7         neighbors ← neigbors ∪ {(G1, M1)}
8      else
9         min ← min \ {g}
10  return neighbors
```

Note that _′ operation denotes computation of intent from extent, or vice versa, implicitly given a context.

Formal concept analysis pseudo-code (2/2)

Software Improvement Group


```
LATTICE (G, M, I)
 1  c ← (∅'', ∅')
 2  insert (c, L)
 3  loop
 4    foreach x in NEIGHBORS (c, (G, M, I))
 5      try x ← lookup (x, L)
 6      with NotFound → insert (x, L)
 7      x* ← x* ∪ {c}
 8      c* ← c* ∪ {x}
 9    try c ← next (c, L)
10    with NotFound → exit
11  return L
```

Transposition to Haskell?

Representation

```
type Context g m = Rel g m
type Concept g m = (Set g, Set m)
type ConceptLattice g m
   = Rel (Concept g m) (Concept g m)
```

Algorithm

```
neighbors :: (Ord g, Ord m)
          => Set g                -- extent of concept
          -> Context g m          -- formal context
          -> [Concept g m]        -- list of neighbors

lattice :: (Ord g, Ord m)
        => Context g m            -- formal context
        -> ConceptLattice g m     -- concept lattice
```

Ongoing projects at SIG

Software Improvement Group

Reverse engineering

- Java library for binary relational algebra (+ slicing + fork)

Repository mining

- Analyze relationships between code/commits/issues through time
- Clustering and decision trees

Quality and metrics

- Generalized method for derivation of quality profiles
- Metrics for architectural quality and evolution
- Benchmarking commercial and open source software

Possible MFES projects at SIG

Java library binary relational algebra

- Extend to “labeled” relations
- Extend with advanced algorithms (e.g. concept analysis)

Randomized testing for Java

- Study existing approaches
- Build / extend tool

More info? Feel free to contact…

Dr. ir. Joost Visser

E: j.visser@sig.nl
W: www.sig.nl
T: +31 20 3140950


Software Improvement Group