

Are you Afraid of Change?

Metrics for Software Evolvability

A historical painting of a Dutch city harbor, likely Delft, featuring a large stone building with a clock tower and a church spire in the background. The scene is set along a canal with several boats and a bridge. The sky is overcast.

Arie van Deursen, Delft University of Technology
Joint work with Eric Bouwers and Joost Visser (SIG)
UC Irvine, March 15, 2013 @avandeursen

View on Delft
Johannes Vermeer
1662







- 2 mile tunnel + station
- 4 train tracks
- Parking for 100 cars
- 1200 new apartments
- 24,000 m2 park
- Parking for 4000 bikes

How would you manage this 15 year 650M Euro project?

The TU Delft Software Engineering Research Group

Education

- Programming, software engineering
- MSc, BSc projects

Research

- Software testing
- Software architecture
- Repository mining
- Collaboration
- End-user programming
- Reactive programming
- Language workbenches



SERG Research Partners

Google™

PHILIPS

Microsoft®

ORACLE®



KPMG



Exact®
software

ROBEKO
The Investment Engineers

SIG
Software Improvement Group

infotron

LOGICBLOX®

Collect detailed ***technical findings***
about software-intensive systems

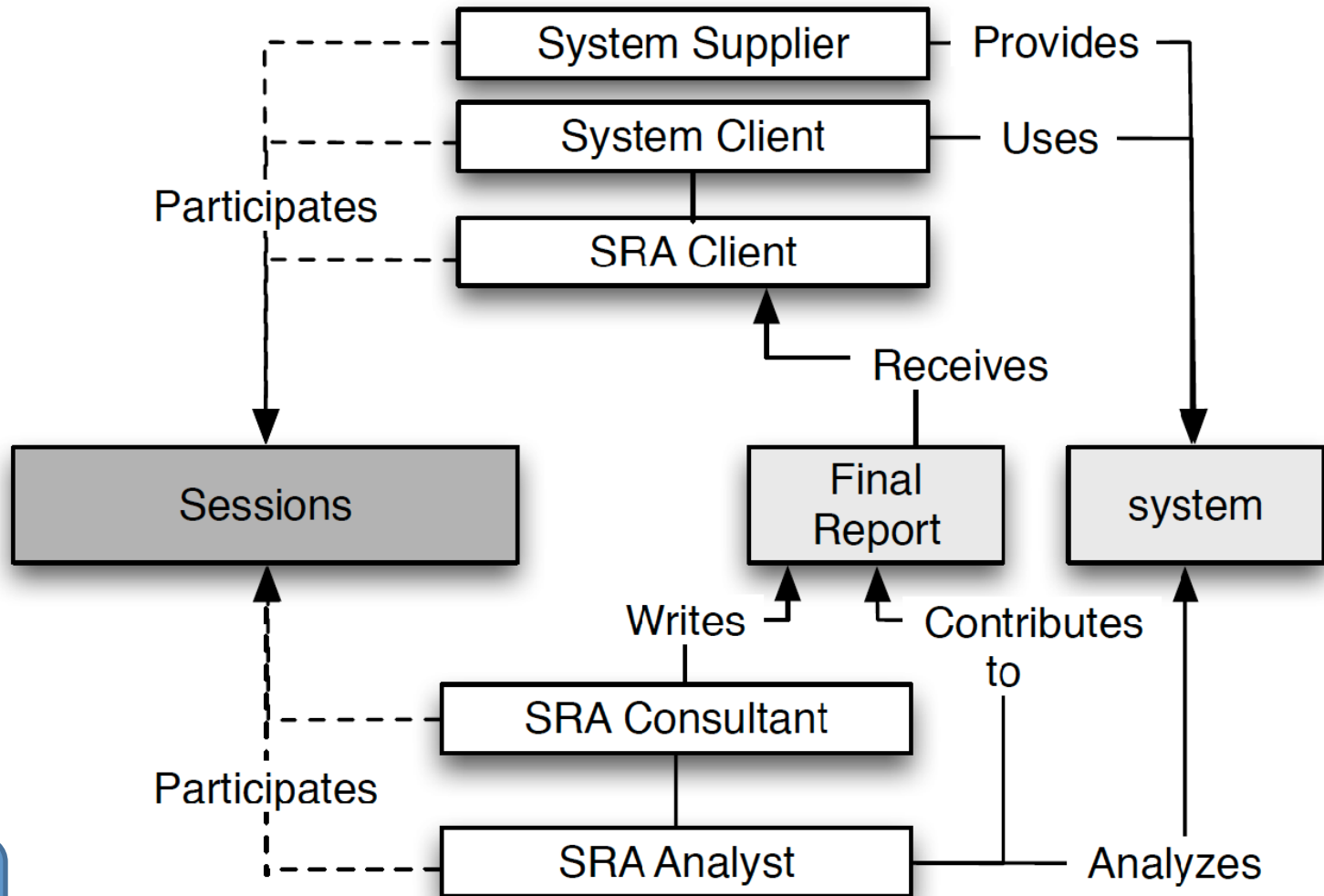
Translate into ***actionable information***
for ***high-level management***

Using methods from academic and
self-funded ***research***

Today's Programme

- **Goal:** Can we measure software quality?
- **Approach:** How can we evaluate metrics?
- **Research:** Can we measure encapsulation?
- **Outlook:** What are the implications?

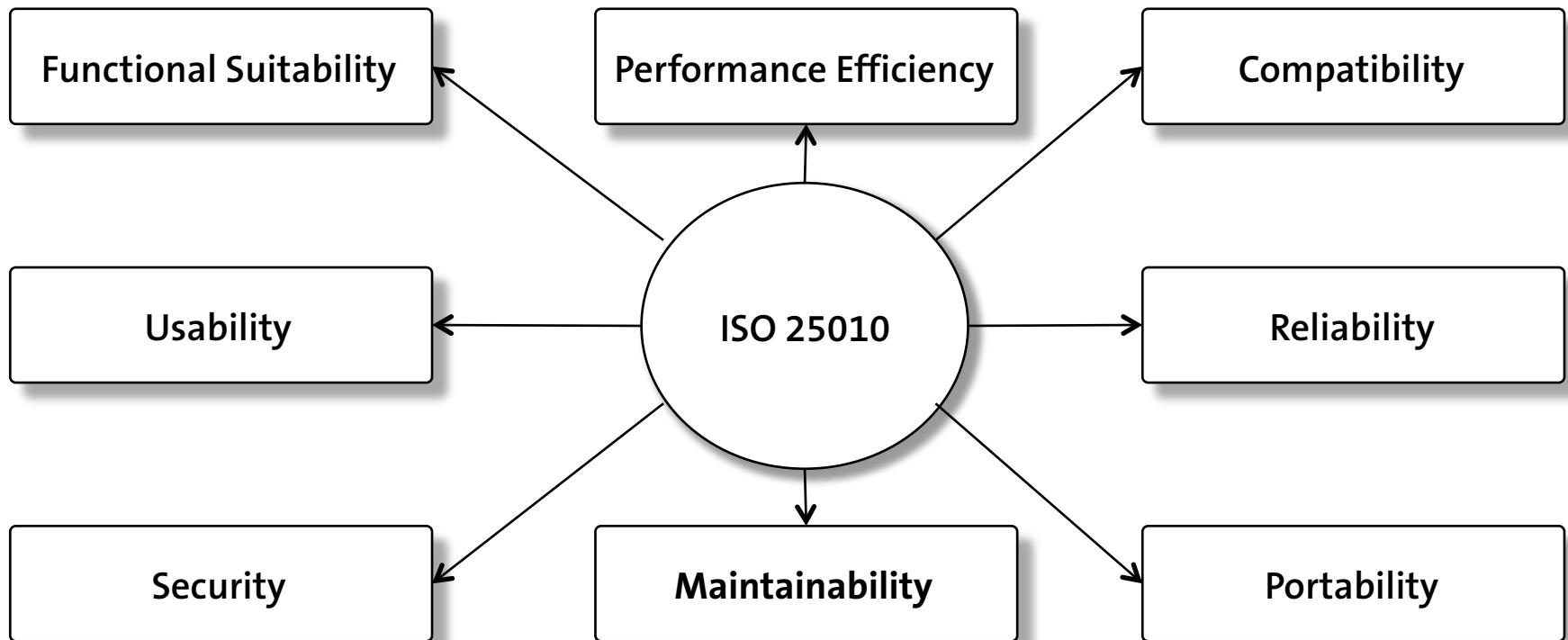
Context: *Software Risk Assessments*



Early versus Late Evaluations

- Today's topic: "Late" evaluations.
 - Actually implemented systems
 - In need of change
- Out of scope today:
 - "Early" evaluation (e.g., ATAM)
 - Software process (improvement)

ISO Software Quality Characteristics



Software Metric Pitfalls

Reflections on decade of metric usage

E. Bouwers, J. Visser, and A. van Deursen. Getting what you Measure. CACM, May 2012

practice

DOI:10.1145/2209249.2209266

Article development led by ACM/queue
queue.acm.org

Four common pitfalls in using software metrics for project management.

BY ERIC BOUWERS, JOOST VISSER, AND ARIE VAN DEURSEN

Getting What You Measure

ARE SOFTWARE METRICS helpful tools or a waste of time?

For every developer who treasures these mathematical abstractions of software systems there is a developer who thinks software metrics are invented just to keep project managers busy. Software metrics can be very powerful tools that help achieve your goals but it is important to use them correctly, as they also have the power to demotivate project teams and steer development in the wrong direction.

For the past 11 years, the Software Improvement Group has advised hundreds of organizations concerning software development and risk

management on the... We have used software investigations in wh... of a system. Addition... track the ongoing de... 400 systems. While... learned some pitfall... metrics in a project... article addresses the

- ▶ Metric in a bubble;
- ▶ Treating the metric;
- ▶ One-track metric; and
- ▶ Metrics galore.

Knowing about these pitfalls will help you recognize them and, hopefully, avoid them, which ultimately leads to making your project successful. As a software engineer, your knowledge of these pitfalls helps you understand why project managers want to use software metrics and helps you assist the managers when they are applying metrics in an inefficient manner. As an outside consultant, you need to take the pitfalls into account when presenting advice and proposing actions. Finally, if you are doing research in the area of software metrics, knowing these pitfalls will help place your new metric in the right context when presenting it to practitioners. Before diving into the pitfalls, let's look at why software metrics can be considered a useful tool.

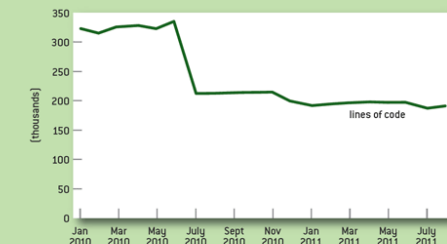
Software Metrics Steer People

"You get what you measure." This phrase definitely applies to software project teams. No matter what you define as a metric, as soon as it is used to evaluate a team, the value of the metric moves toward the desired value. Thus, to reach a particular goal, you can continuously measure properties of the desired goal and plot these measurements in a place visible to the team. Ideally, the desired goal is plotted alongside the current measurement to indicate the distance to the goal.

Imagine a project in which the run-time performance of a particular use

FIGURE 1

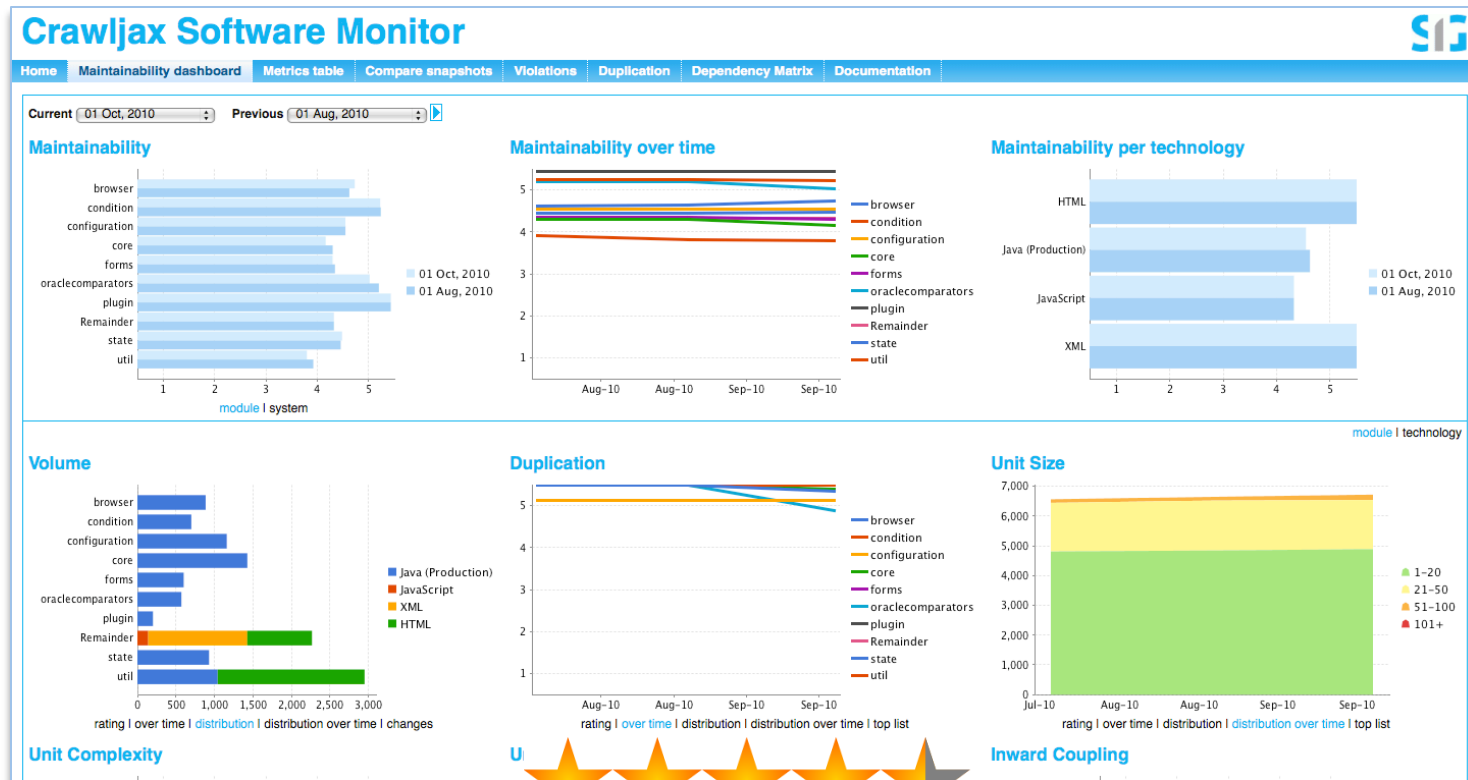
Lines of Code of a Software System



42 COMMUNICATIONS OF THE ACM

Pitfall 1: Treating the Metric

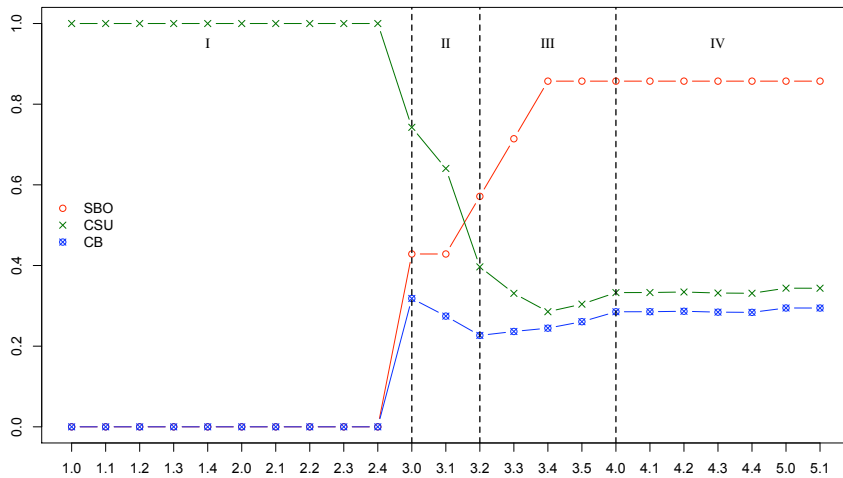
Metric values are symptoms:
It's the root cause that should be addressed



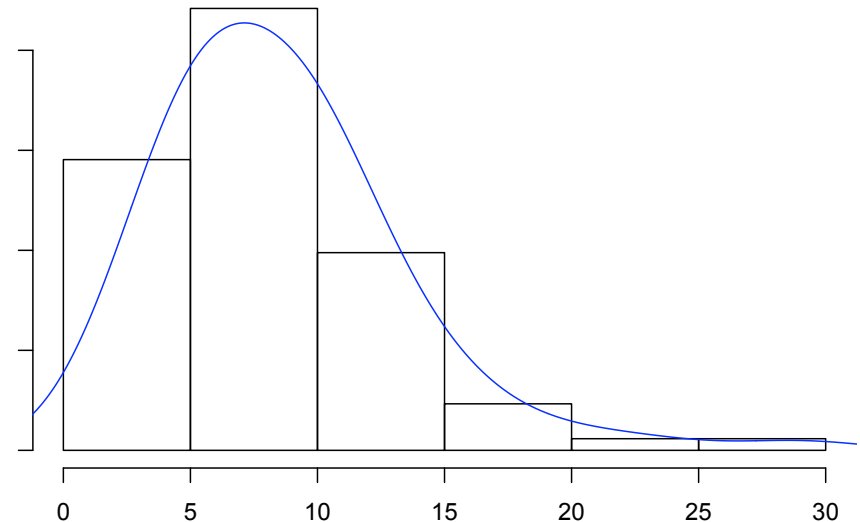
Pitfall 2: Metric in a Bubble

To interpret a metric, a *context* is needed

Temporal / Trend

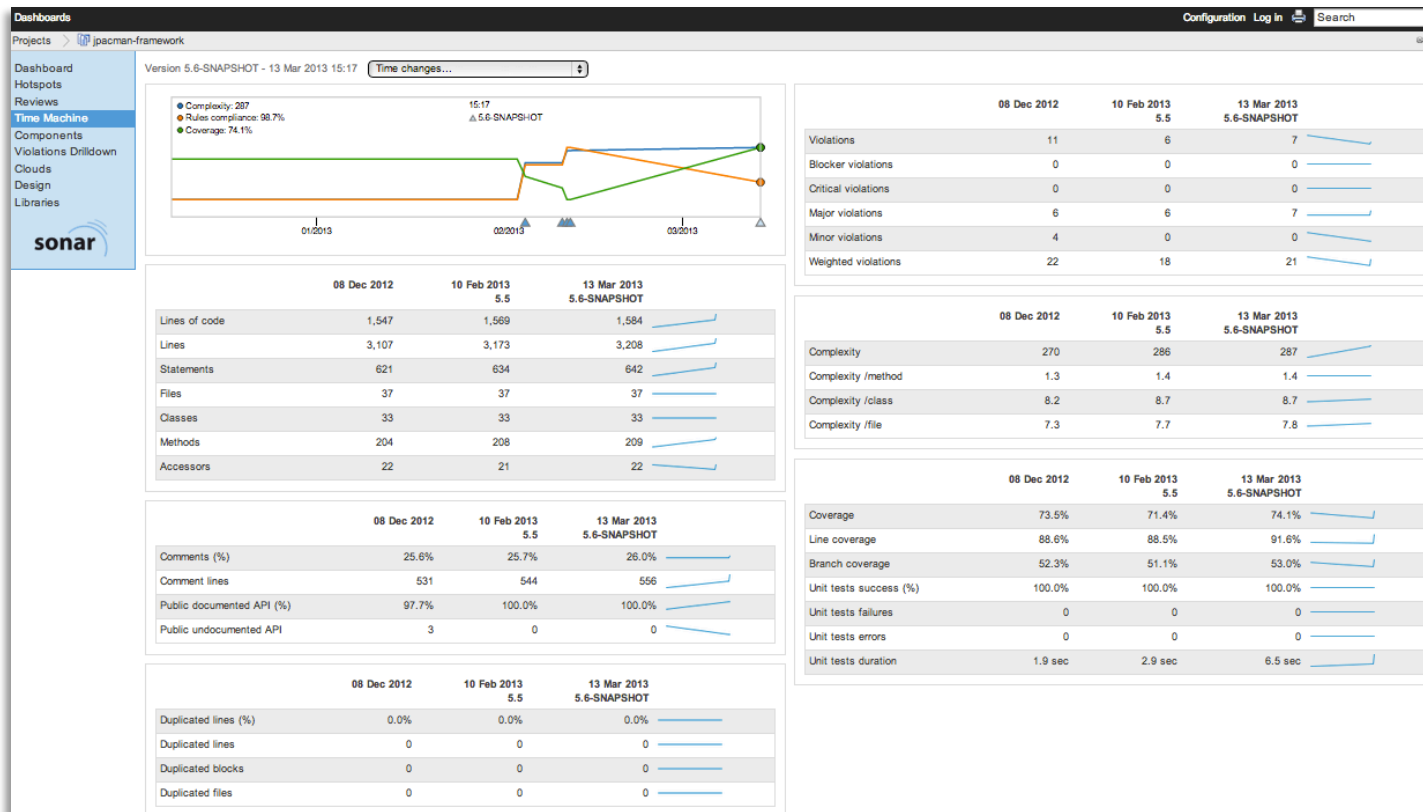


Peers / Norms



Pitfall 3: Metrics Galore

Not everything that can be measured
needs to be measured



Pitfall 4: One Track Metric

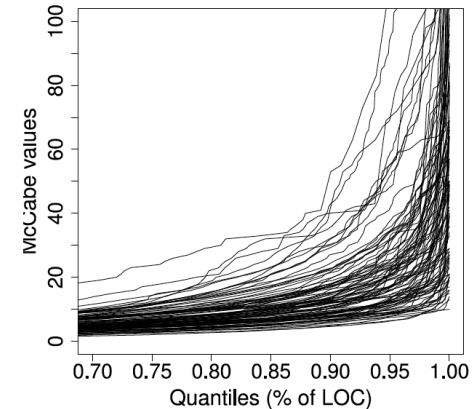
Trade-offs in design require multiple metrics

In carefully crafted metrics suite,
negative side effects of
optimizing one metric
are counter-balanced
by other ones



Putting Metrics in Context

- Establish benchmark
 - Range of industrial systems with metric values
- Determine thresholds based on quantiles.
 - E.g.: 70%, 80%, 90% of systems
 - No normal distribution



Example: McCabe.
90% of systems have average unit complexity that is below 15.

Tiago L. Alves, Christiaan Ypma, Joost Visser.
Deriving metric thresholds from benchmark data. *ICSM* 2010.

Assessments 2003--2008

- ISO 9126 quality model
- ~50 assessments
- Code/module level metrics
- Architecture analysis always included
 - *No architectural metrics used.*

“Architectures allow or preclude nearly all of a system’s quality attributes.”

-- Clements et al, 2005

Heitlager, Kuipers, Visser. A Practical Model for Measuring Maintainability. QUATIC 2007

Van Deursen, Kuipers. Source-Based Software Risk Assessments, ICSM 2003

2009: Re-thinking Architectural Analysis

Qualitative study of
40 risk assessments

Which architectural
properties?

Outcome: Metrics
refinement wanted

	High Level Design	Modularization	Separation of Concerns
Abstraction	8	3	2
Functional Duplication	2	6	18
Layering	28	1	20
Libraries / Frameworks	22	1	1
Logic in Database	1	1	3
Module Dependencies	7	11	6
Module Functionality	4	32	13
Module Inconsistency	0	1	0
Module Size	1	1	0
Relation Documentation / Implementation	2	3	0
Source Grouping	0	14	2
Technology Age	13	0	0
Technology Usage	7	3	0
Technology Combination	5	1	0
Textual Duplication	0	0	4

Eric Bouwers, Joost Visser, Arie van Deursen:
Criteria for the evaluation of implemented architectures. ICSM 2009

ISO 25010 Maintainability

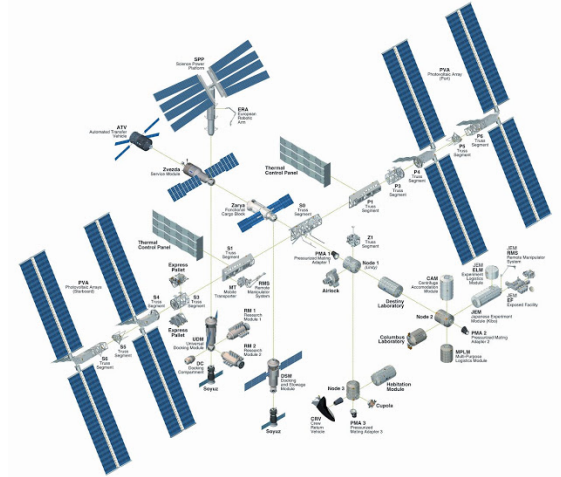
“Degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers”

Five sub-characteristics:

- Analyzability, Modifiability,
- Testability, Reusability
- Modularity

Modularity

ISO 25010 maintainability sub characteristic:



“Degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components”

Information Hiding



Things that change at the same rate belong together.

Things that change quickly should be insulated from things that change slowly.

Kent Beck. Naming From the Outside In.
Facebook Blog Post, September 6, 2012.

Measuring Encapsulation?



Can we find software architecture metrics that can serve as indicators for the success of encapsulation of an implemented software architecture?

Eric Bouwers, Arie van Deursen, and Joost Visser.

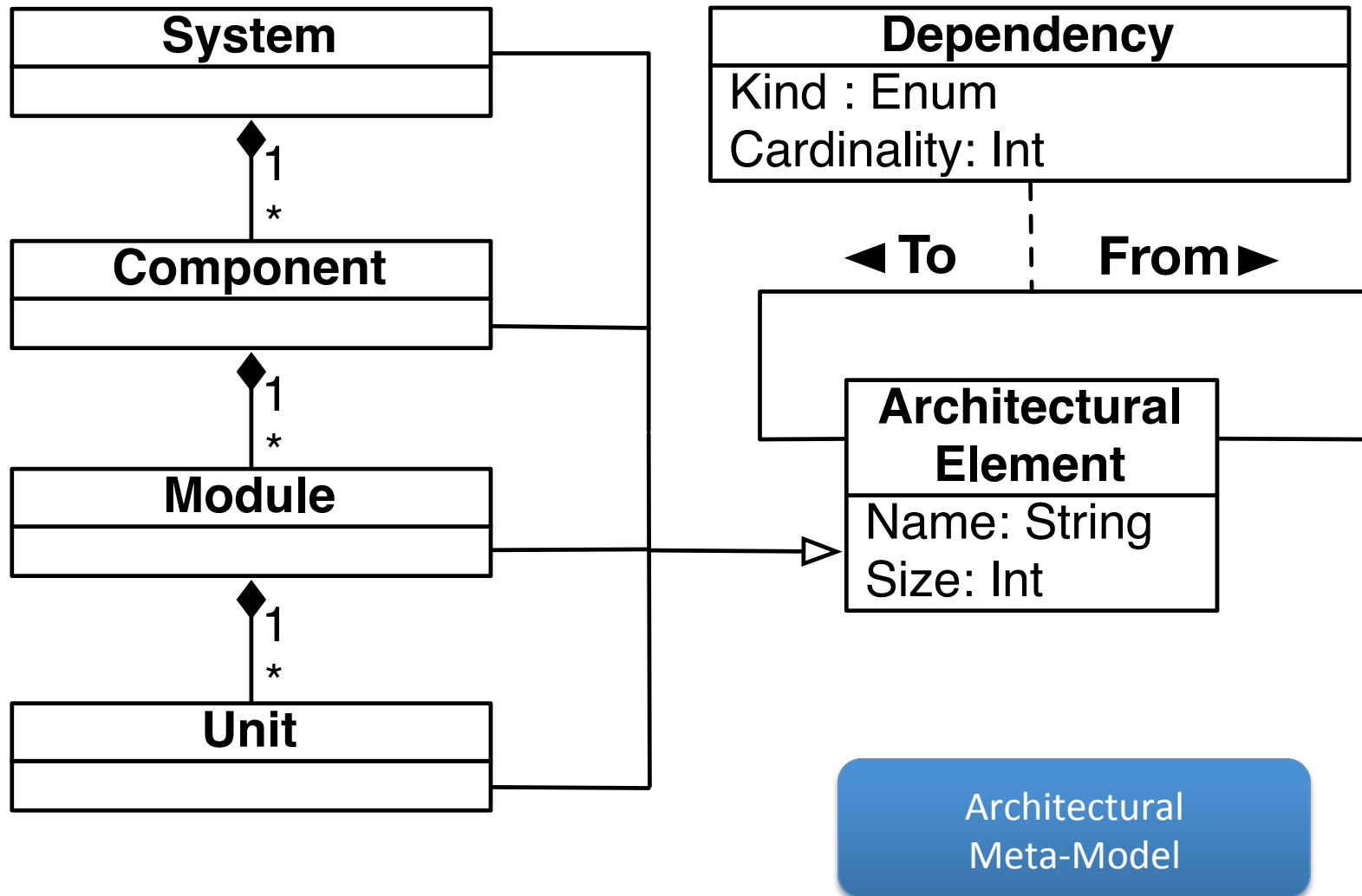
Quantifying the Encapsulation of Implemented Software Architectures

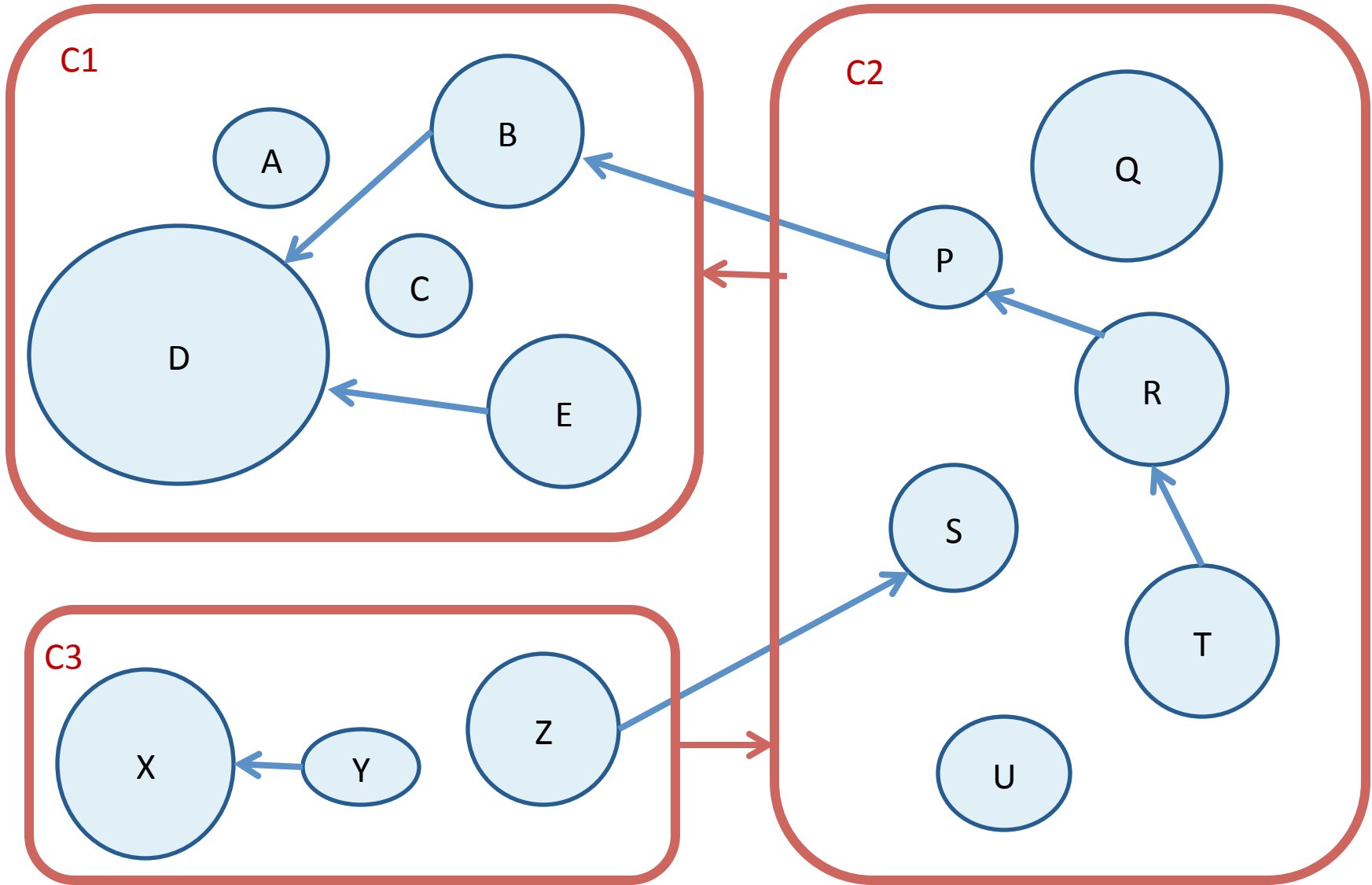
Technical Report TUD-SERG-2011-031-a, Delft University of Technology, 2012

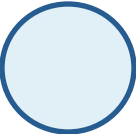
Metric Criteria in an Assessment Context

1. Potential to measure the level of encapsulation within a system
2. Is defined at (or can be lifted to) the system level
3. Is easy to compute and implement
4. Is as independent of technology as possible
5. Allows for root-cause analysis
6. Is not influenced by the volume of the system under evaluation


What is an Architecture?





 Module
 (size)

 Component

 Module dependency

 Lifted (comp) dependency

Searching the Literature

- Identified over 40 candidate metrics
- Survey by Kozirolek starting point
- 11 metrics meet criteria

Sustainability Evaluation of Software Architectures: A Systematic Review

Heiko Kozirolek¹

¹Industrial Software Systems, ABB Corporate Research, Ladenburg, Germany
heiko.kozirolek@de.abb.com

ABSTRACT

Long-living software systems are sustainable if they can be cost-efficiently maintained and evolved over their entire life-cycle. The quality of software architectures determines sustainability to a large extent. Scenario-based software architecture evaluation methods can support sustainability analysis, but they are still reluctantly used in practice. They are also not integrated with architecture-level metrics when evaluating implemented systems, which limits their capabilities. Existing literature reviews for architecture evaluation focus on scenario-based methods, but do not provide a critical reflection of the applicability of such methods for sustainability evaluation. Our goal is to measure the sustainability of a software architecture both during early design using scenarios and during evolution using scenarios and metrics, which is highly relevant in practice. We thus provide a systematic literature review assessing scenario-based methods for sustainability support and categorize more than 40 architecture-level metrics according to several design principles. Our review identifies a need for further empirical research, for the integration of existing methods, and for the more efficient use of formal architectural models.

1. INTRODUCTION

Software systems with a life span of more than 15 years must be designed and implemented carefully so that they are prepared for maintenance and evolution. During their life-time such systems inevitably undergo many corrective, adaptive, enhance, and preventive changes. This is especially pronounced in the industrial automation domain, where software systems are embedded in complex technical hardware/software environments. Software architectures are a major driver for the sustainability (i.e., cost-efficient longevity) and evolvability [12, 73], because they influence how quickly and correctly a developer is able to understand, analyse, extend, test, and maintain a software system. Evaluating and improving the sustainability of a software architecture is thus a major concern for software architects.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

While researchers have proposed many scenario-based evaluation methods [25], it is not well understood how they support improving the sustainability of a system. In practice many architects still mainly rely on experience and prototyping to support their design decisions [10]. For implemented architectures, architecture-level code metrics assessing modularization quality can add valuable information to a sustainability evaluation [18], but an overview and systematic validation of such metrics is missing. Thereby, architecture-level code metrics are still sparsely used in practice.

Existing literature reviews for architecture evaluation methods [26, 9, 38, 11] focus mainly on scenario-based methods to evaluate early software architecture designs and do not analyse their suitability for sustainability evaluation. Other surveys [11, 59, 19] provide more breadth but do not include architecture-level metrics either. Reviews of architecture-level metrics cannot be found in literature, as related studies (e.g., [57]) focus on class-level OO metrics (e.g., McCabe [49], Halstead [83], Chidamber [24]) and neglect metrics for higher-level code structures.

The contribution of this paper is a structured literature review on methods and metrics for evaluating the sustainability of software architectures. Our review carefully analyses existing scenario-based methods for their suitability to evaluate sustainability and additionally provides a survey and analysis of more than 40 architecture-level metrics. An integration of scenario-based and metrics-based methods is useful to provide a continuous, pro-active approach towards evolution problem throughout the entire system life-cycle. Our survey is intended to help practitioners to select a method reflecting their specific requirements, and to help researchers to identify gaps and pointers for future work in the existing body of work. Our review also provides the base for a possible integration of both kinds of methods in a combined and even more valuable approach.

The remainder of this paper is as follows [39]: Section 2 defines the most important terms and motivates the need for a new review. Section 3 states our research questions, list the data sources, inclusion criteria and data collections. Section 4 then presents the results of the review, which shall answer the formally stated research questions. Section 5 discusses the results and provides implications for research and practice. Finally, Section 6 concludes the paper.

2. BACKGROUND

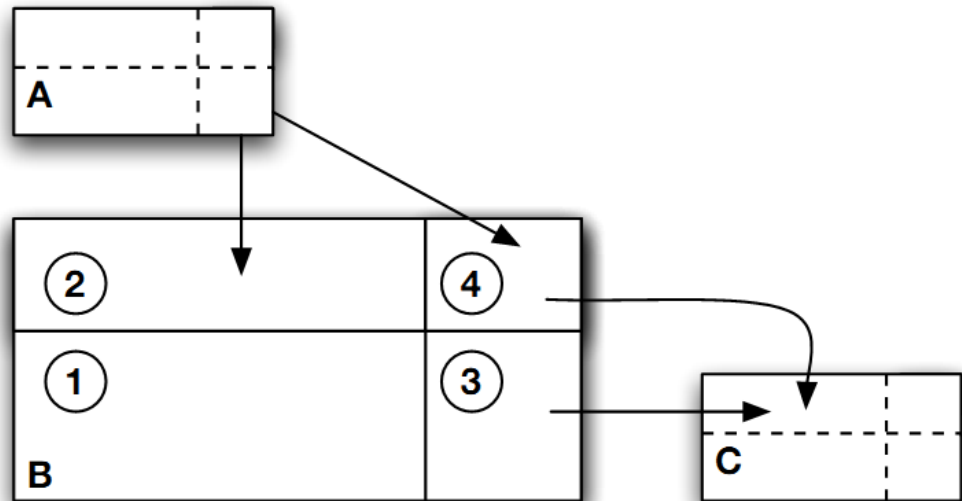
This section first defines the terms 'sustainability' (Section 2.1) and 'sustainability' (Section 2.2) and then (Section 2.3).

H. Kozirolek. Sustainability evaluation of software architectures: a systematic review. In QoSA-ISARCS '11, pages 3–12. ACM, 2011

Our own Proposal: Dependency Profiles

Module types:

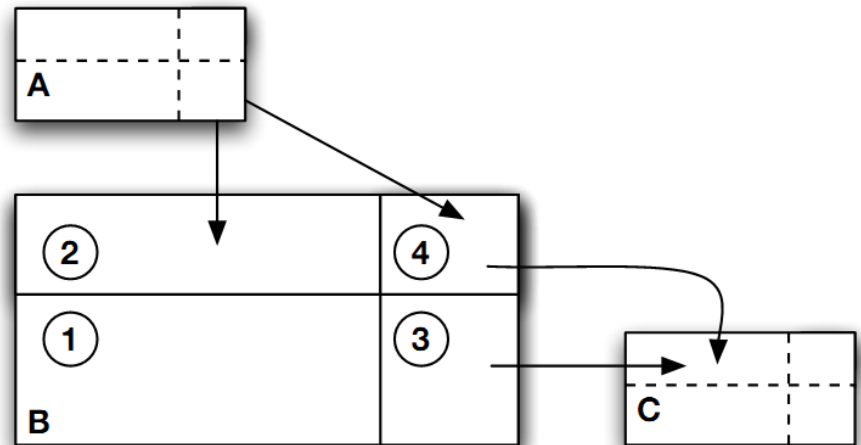
1. Internal
2. Inbound
3. Outbound
4. Transit



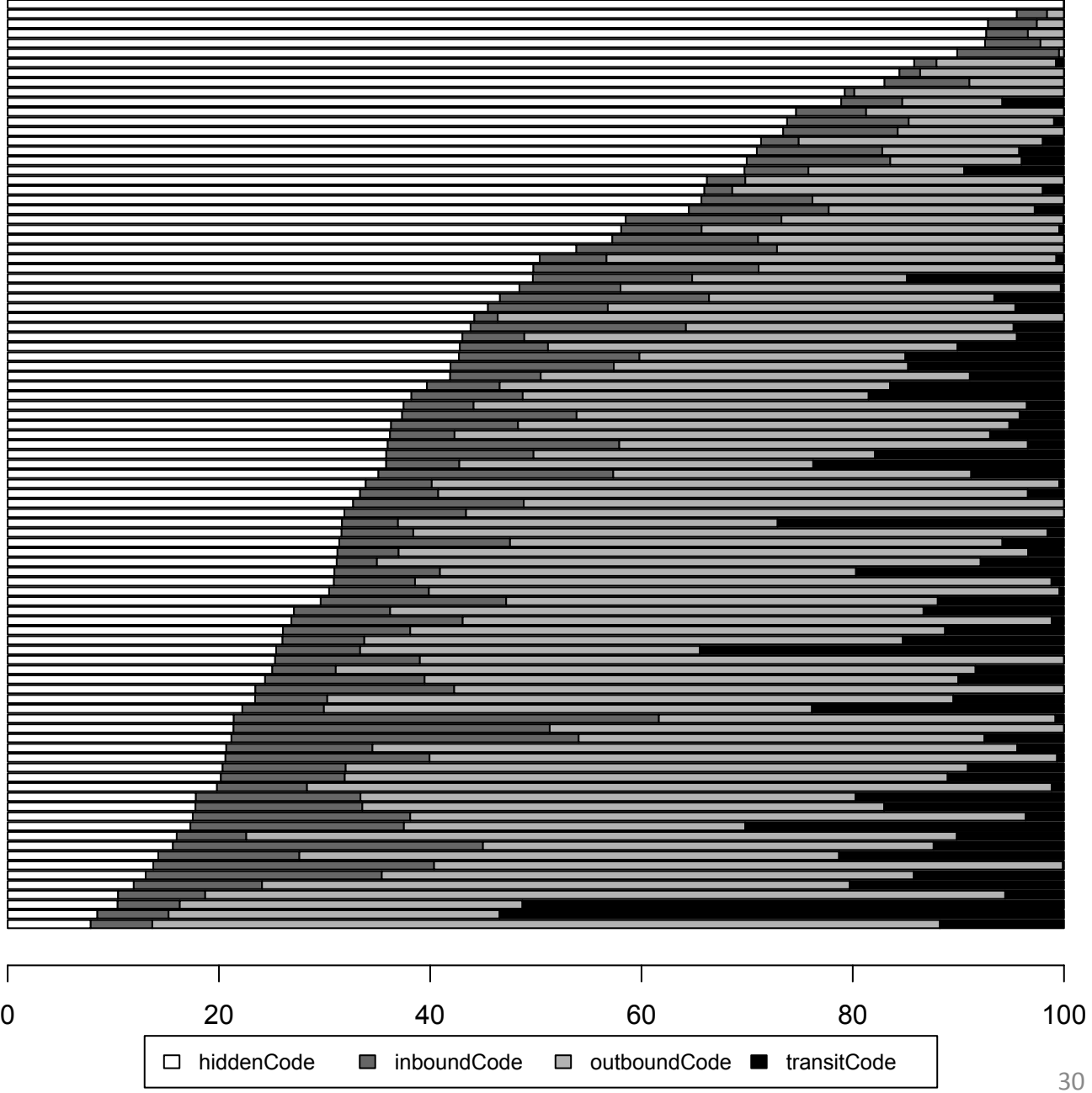
Eric Bouwers, Arie van Deursen, Joost Visser.
Dependency Profiles for Software Architecture Evaluations. ICSM ERA, 2011.

Dependency Profiles (2)

- Look at relative *size* of different module types
- Dependency profile is quadruple:
<%internal, %inbound, %outbound, %transfer>
- <40, 30, 20, 10> versus <60, 20, 10, 0>
- Summary of componentization at the system level



Profiles in benchmark of ~100 systems



Literature Study: Candidate Metrics

Name	Abbr.	Src.
Ratio of Cohesive Interactions	RCI	[Briand et al. 1993]
Cumulative Component Dependency	CCD	[Lakos 1996]
Average CCD	ACD	[Lakos 1996]
Normalized CCD	NCD	[Lakos 1996]
Cyclic Dependency Index	CDI	[Sarkar et al. 2007]
Inbound code	IBC	[Bouwers et al. 2011b]
Outbound code	OBC	[Bouwers et al. 2011b]
Internal code	IC	[Bouwers et al. 2011b]
Number of Binary Dependencies	NBD	
Component Balance	CB	[Bouwers et al. 2011a]
Module Size Uniformity Index	MSUI	[Sarkar et al. 2007]
Number of components	NC	

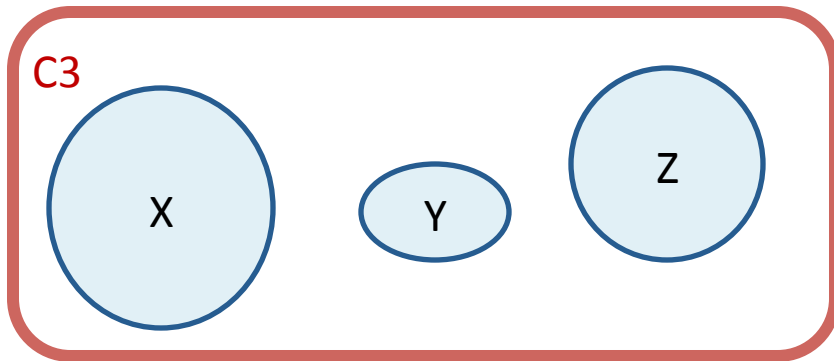
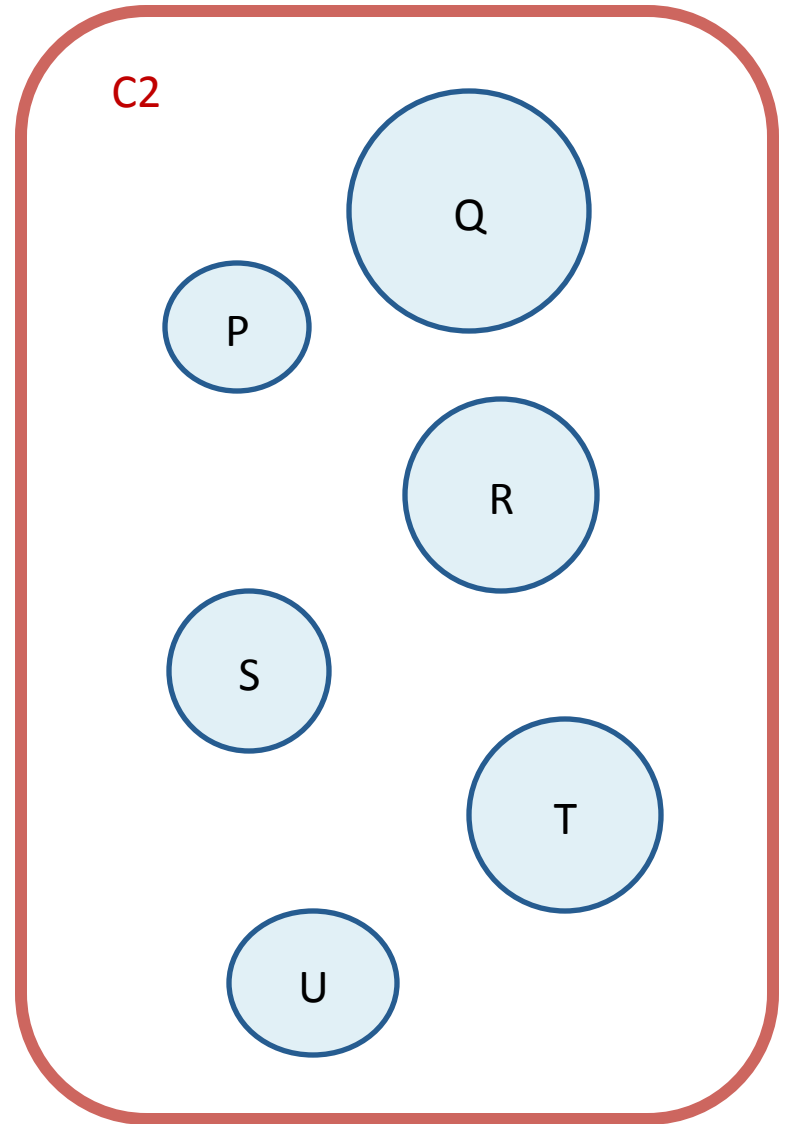
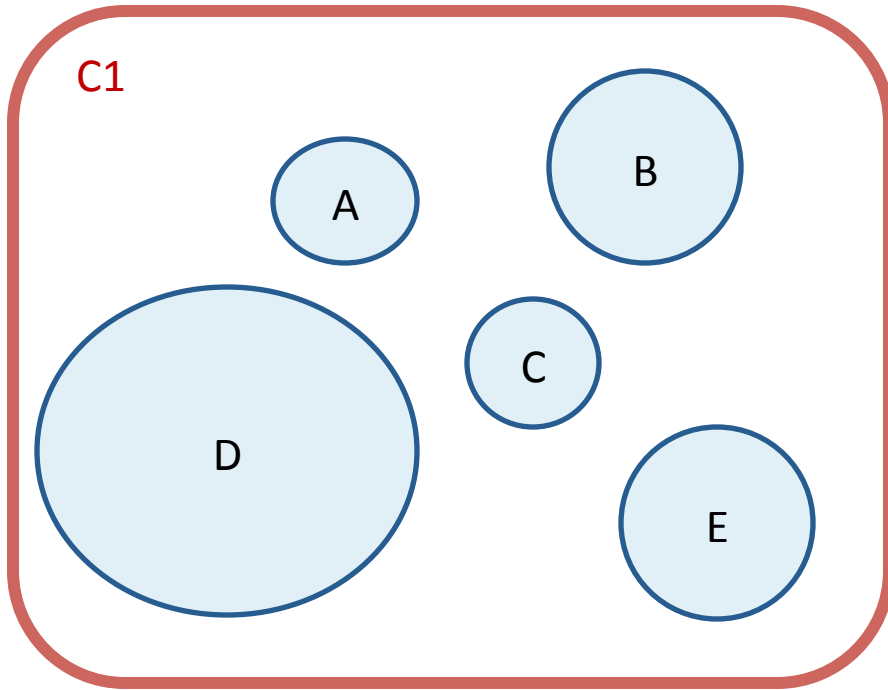
Metrics Evaluation

1. Quantitative approach

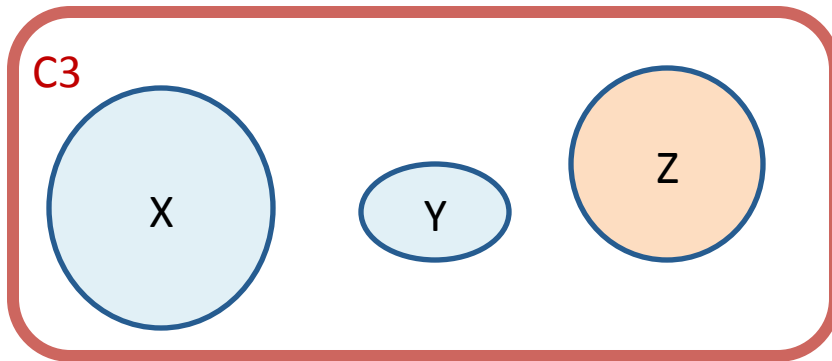
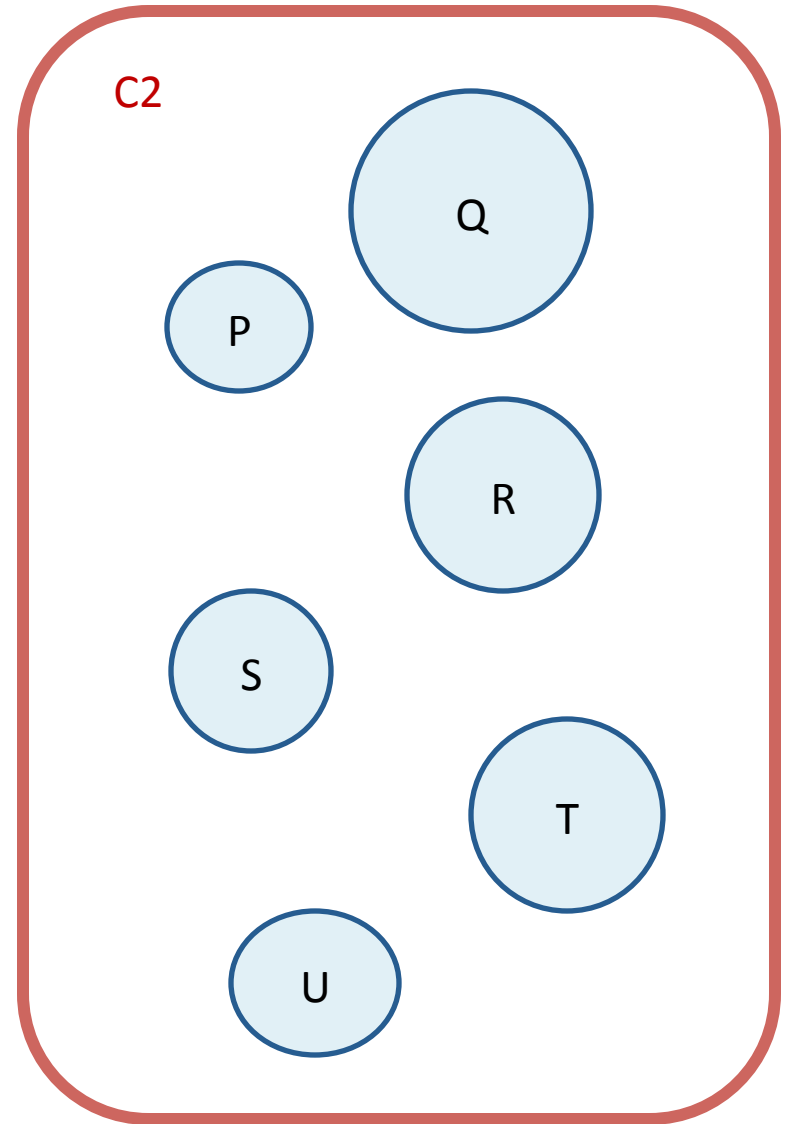
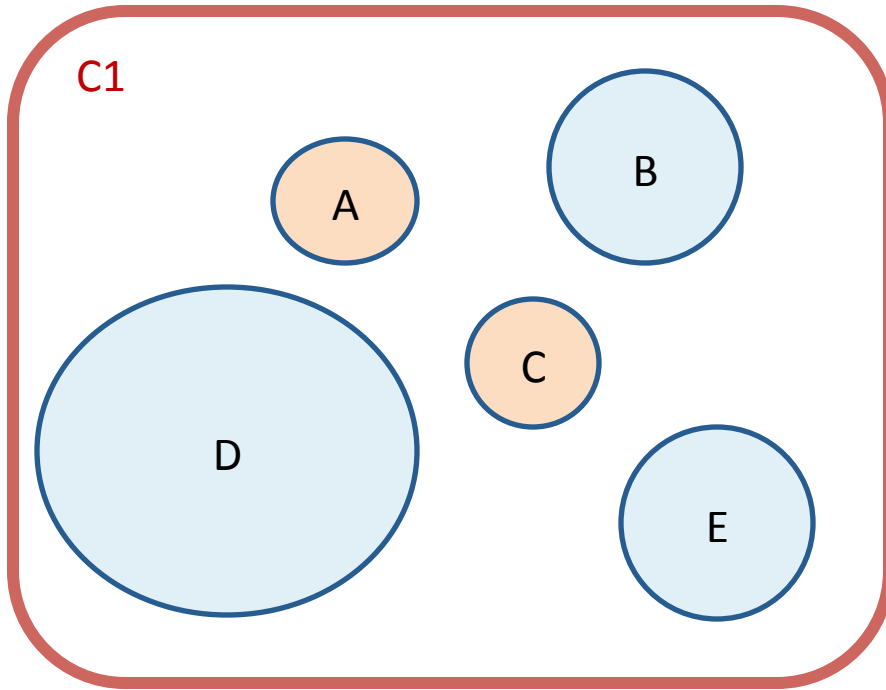
- *Which metric is the best predictor of good encapsulation?*
- Compare to *change sets* (repository mining)

2. Qualitative approach:

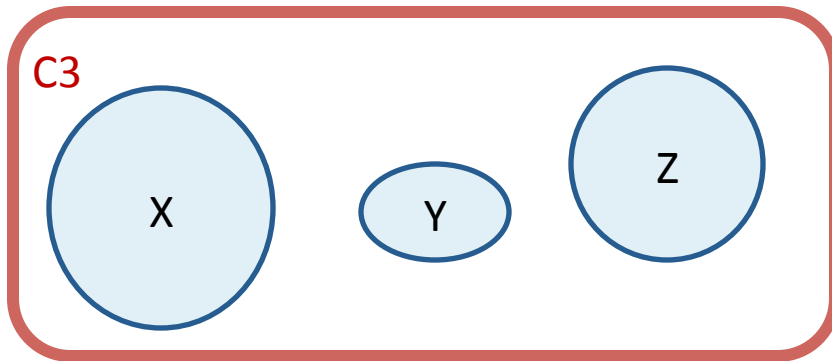
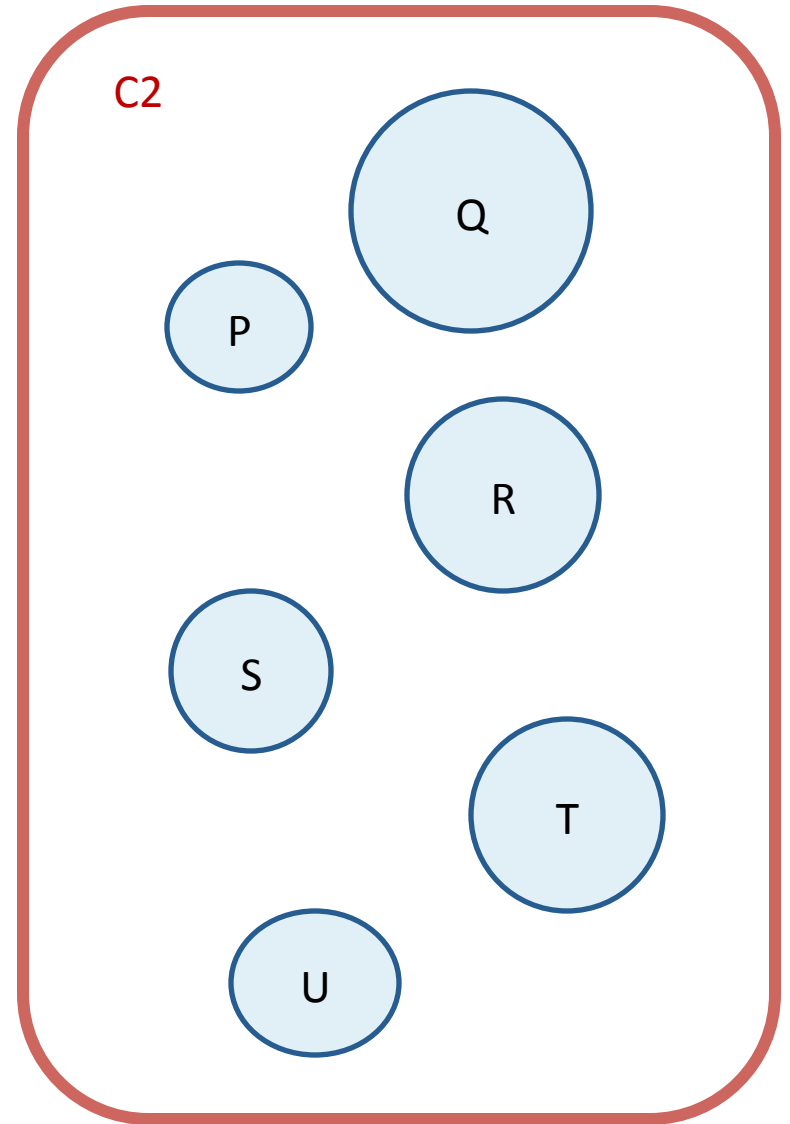
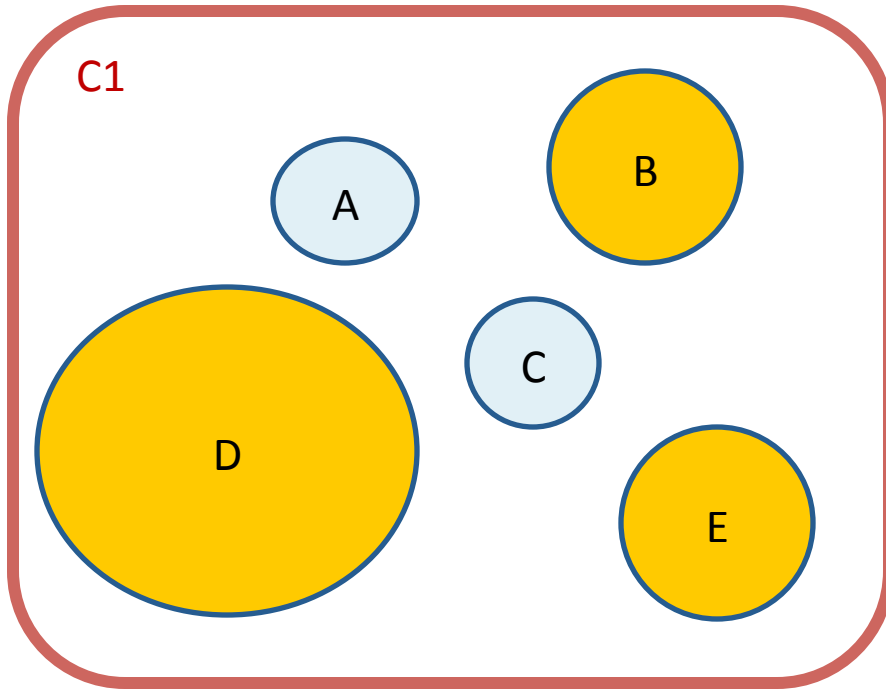
- *Is the selected metric useful in a late architecture evaluation context?*



Commit in version repository results in *change set*

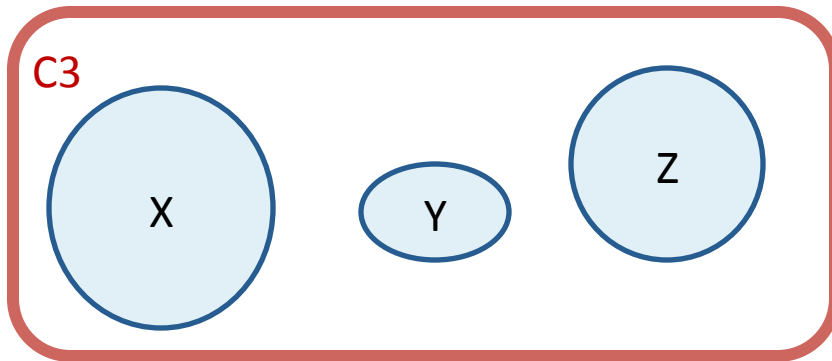
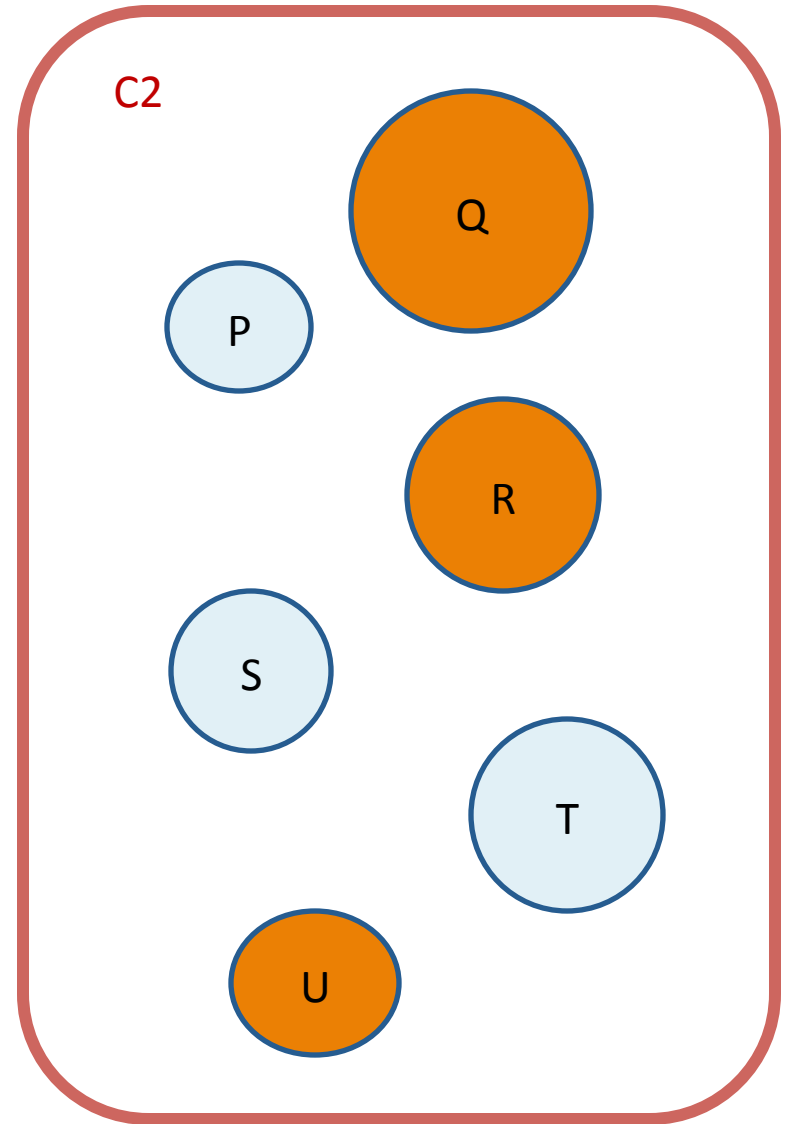
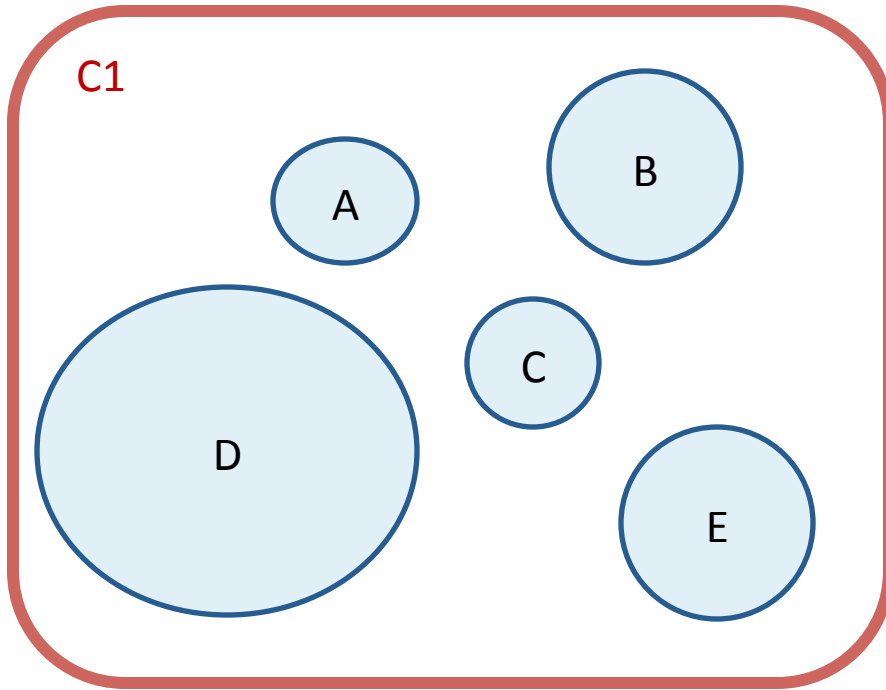


Change set I: modules { A, C, Z }
Affects components C1 and C3



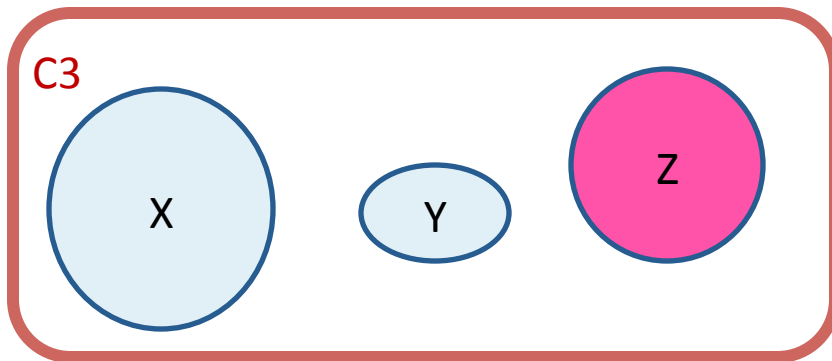
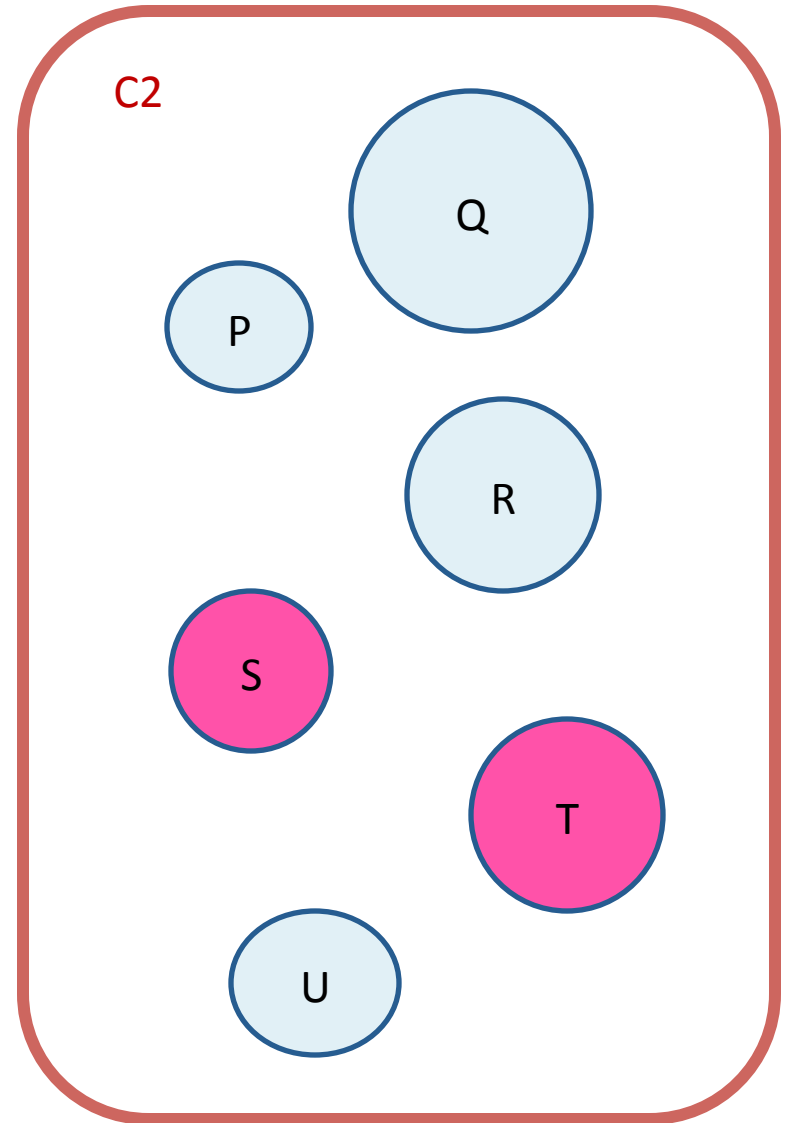
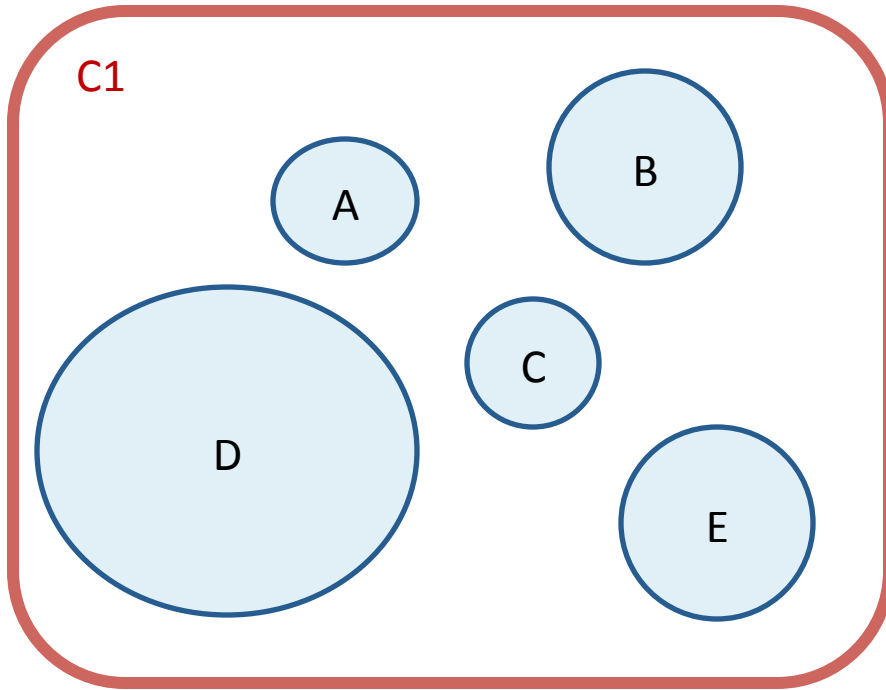
Change set II: modules { B, D, E }
Affects components C1 *only*

Local change



Change set III: modules { Q, R, U }
Affects components C2 *only*

Local change



Change set IV: modules { S, T, Z }
Affects components C2 and C3

Non-Local change₃₇

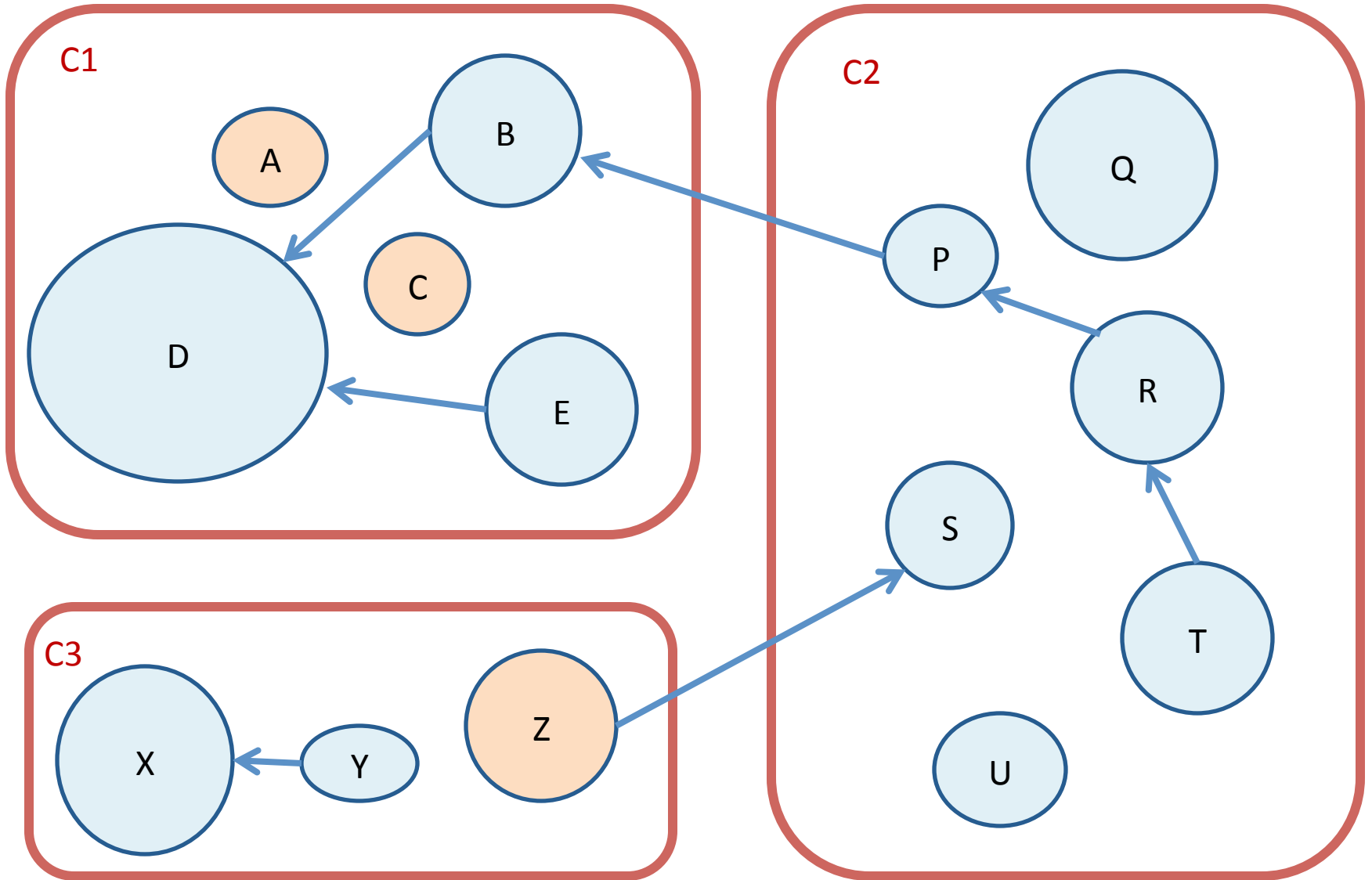
Observation 1: Local Change-Sets are Good

- Combine change sets into *series*
- *The more local changes in a series, the better the encapsulation worked out.*

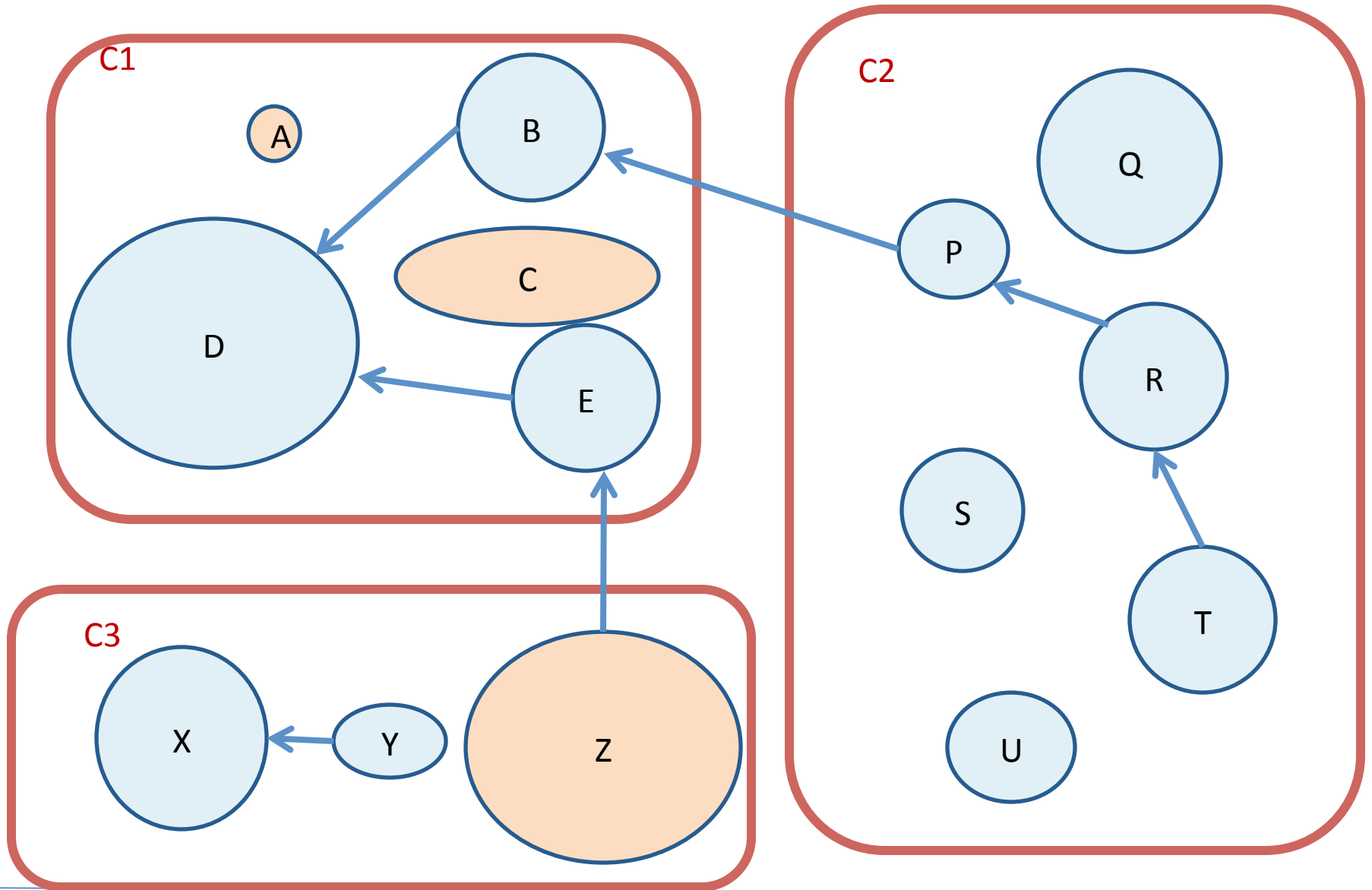
Observation 2:

Metrics may change too

- A change may affect the value of the metrics.
- Cut large set of change sets into sequence of stable change-set series.

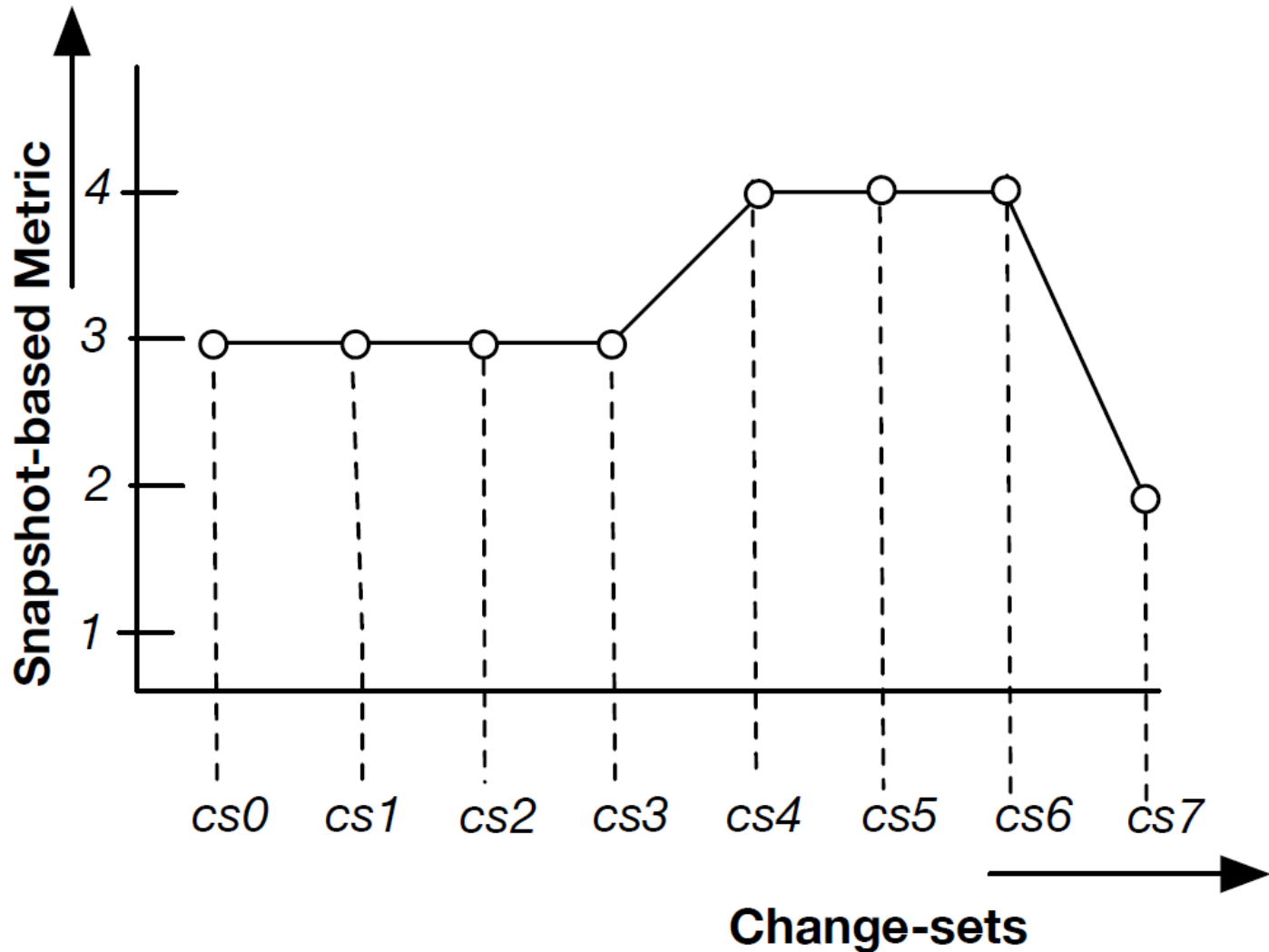


Change set I: modules { A, C, Z }
Affects components C1 and C3



Change set I: modules { A, C, Z }
The Change Set may affect metric outcomes!!

Solution: *Stable Period Identification*



Experimental Setup

- Identify 10 long running open source systems
- Determine metrics on monthly snapshots
- Determine stable periods per metric:
 - Metric value
 - *Ratio of local change* in this period
- Compute (Spearman) correlations [0, .30, .50, 1]
- Assess significance ($p < 0.01$)
- [Assess project impact]
- Interpret results

Systems Under Study

Name	Period		Size (KLOC)	
	Start	End	Start	End
Ant	2000-02	2011-05	3	97
Argouml	2008-03	2011-07	113	108
Beehive	2004-08	2008-10	45	86
Crawljax	2010-01	2011-07	6	7
Findbugs	2003-04	2011-07	7	97
Jasperreports	2004-01	2011-08	28	171
Jedit	2001-10	2011-08	35	79
Jhotdraw	2001-03	2005-05	8	20
Lucene	2001-10	2011-08	6	67
Struts2	2006-06	2011-07	25	22

Stable Periods

Metric	periods	Months				change-sets series length				
		Min	Med.	Max	covered	Min	Med.	Max	total	> 10
RCI	94	1	4.0	38	80.9 %	3	113.0	968	17760	93.6 %
CCD	71	1	6.0	40	85.9 %	3	222.0	1178	19011	97.2 %
ACD	111	1	3.0	38	75.6 %	1	92.0	954	16564	91.9 %
NCD	74	1	4.5	40	83.6 %	3	192.5	1174	17922	95.9 %
CDI	65	1	6.0	50	88.3 %	1	224.0	2334	20526	95.4 %
IBC	122	1	3.0	35	68.1 %	3	67.5	715	13811	95.9 %
OBC	111	1	3.0	42	71.8 %	3	68.0	1337	15346	94.6 %
IC	119	1	2.0	41	71.2 %	2	50.0	1257	14759	91.6 %
NBD	108	1	3.0	38	75.8 %	3	88.5	846	15436	94.4 %
CB	82	1	3.0	77	80.6 %	3	76.5	5147	19345	91.5 %
MSUI	99	1	3.0	35	77.1 %	1	91.0	1176	18028	93.9 %
NC	59	1	6.0	53	90.8 %	7	262.0	1805	21428	96.6 %

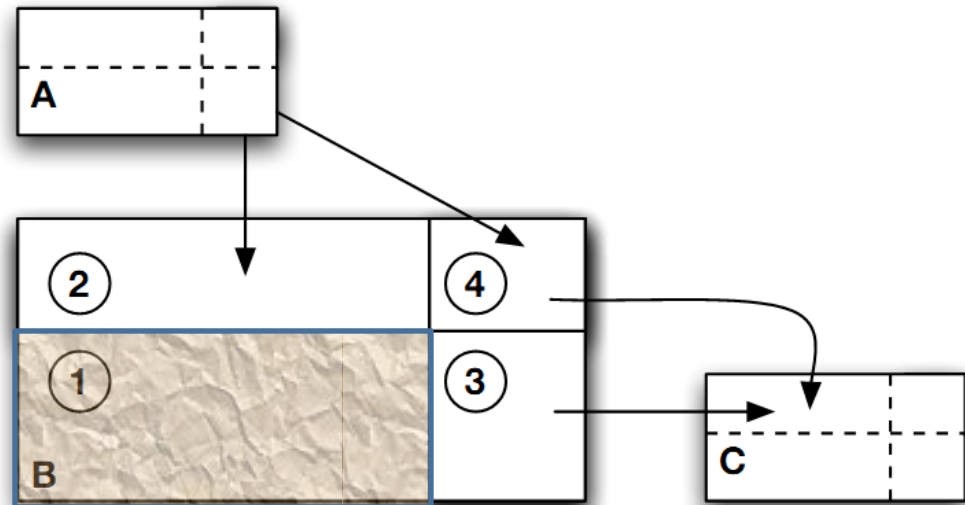
Results

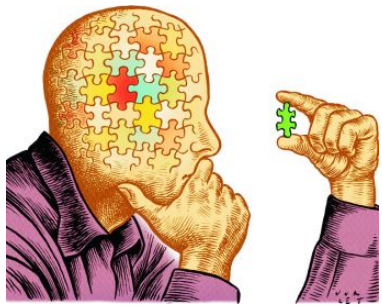
Metric	Correlation	Corrected p-value	p-value
RCI	0.16	11.3	0.94
CCD	-0.27	0.13	0.01
ACD	-0.26	0.04	< 0.01
NCD	-0.19	0.59	0.05
CDI	0.32	11.94	1.00
IBC	-0.30	< 0.01	< 0.01
OBC	-0.31	< 0.01	< 0.01
IC	0.47	< 0.01	< 0.01
NBD	-0.22	0.14	0.01
CB	0.29	0.05	< 0.01
MSUI	-0.08	2.42	0.20
NC	-0.26	0.27	0.02

Best Indicator for Encapsulation: *Percentage of Internal Code*

Module types:

1. Internal
2. Inbound
3. Outbound
4. Transit





Threats to Validity

Construct validity

- Encapsulation == local change?
- Commit == coherent?
- Commit size?
- Architectural model?

Reliability

- Open source systems
- All data available

Internal validity

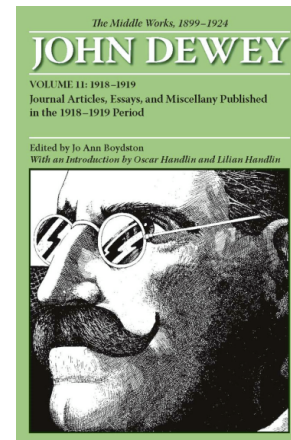
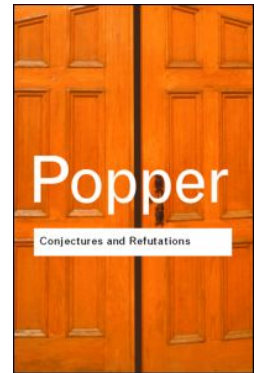
- Stable periods: Length, nr, volume
- Monthly snapshots
- Project factors

External validity

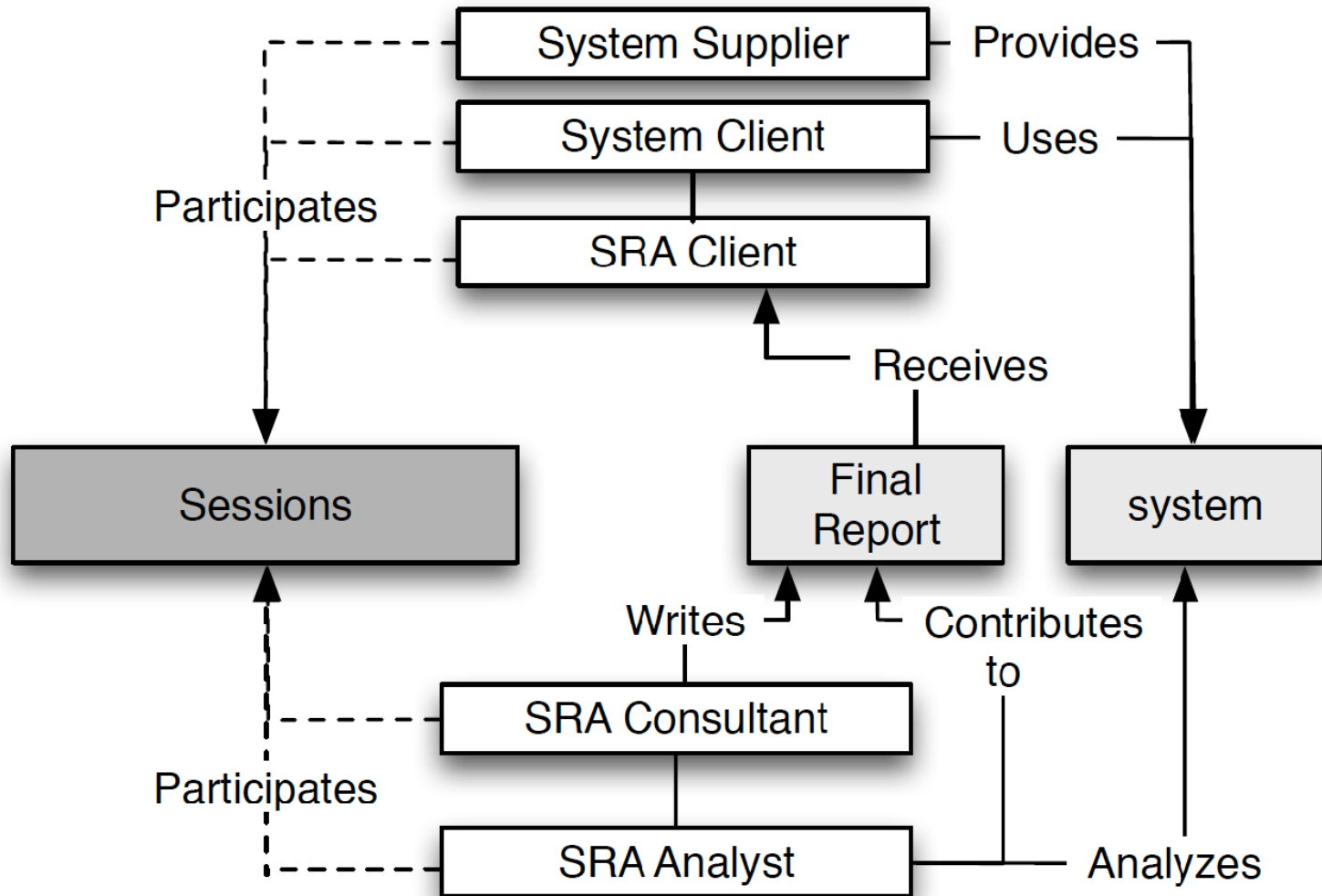
- Open source, Java
- IC behaves same on other technologies

Shifting paradigms

- Statistical hypothesis testing:
Percentage of internal change is valid indicator for encapsulation
- But is it of any *use*?
- Can people work with?
- Shift to pragmatic knowledge paradigm



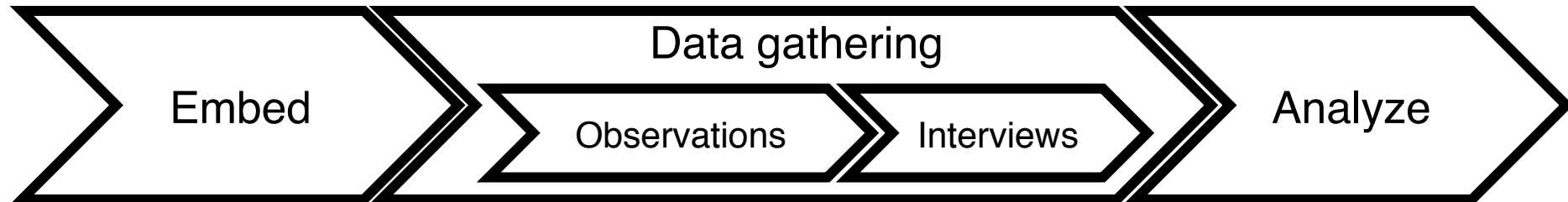
Software Risk Assessments



Experimental Design

Goal:

- Understand the usefulness of dependency profiles
- From the point of view of external quality assessors
- In the context of external assessments of implemented architectures



Eric Bouwers, Arie van Deursen, Joost Visser. *Evaluating Usefulness of Software Metrics; An Industrial Experience Report*. ICSE SEIP 2013

Embedding

- January 2012: New metrics in SIG models
 - 50 risk assessments during 6 months
 - Monitors for over 500 systems
 - “Component Independence”
- System characteristics:
 - C#, Java, ASP, SQL, Cobol, Tandem, ...
 - 1000s to several millions of lines of code
 - Banking, government, insurance, logistics, ...

Data Gathering: Observations

- February-August 2012
- Observer collects stories of actual usage
- Written down in short memos.
- 17 different consultants involved
- 49 memos collected.
- 11 different customers and suppliers

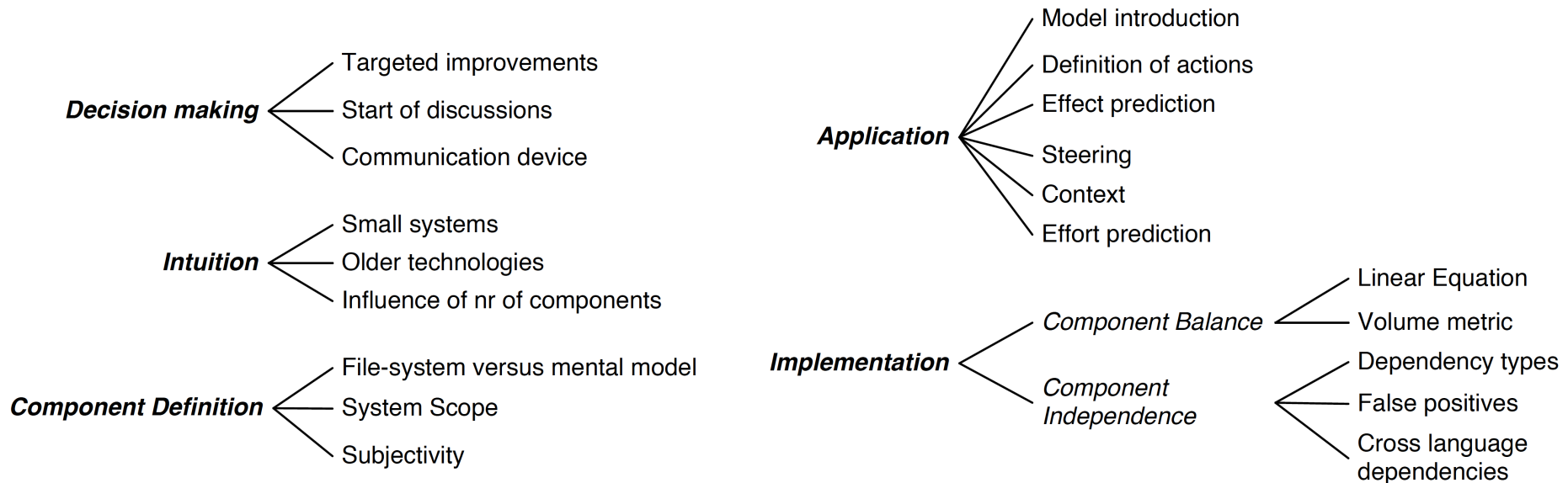


Data Gathering: Interviews

- 30 minute interviews with 11 assessors
- Open discussion:
 - “How do you use the new component independence metric”?
 - Findings in 1 page summaries
- Scale 1-5 answer:
 - How useful do you find the metric?
 - Does it make your job easier?

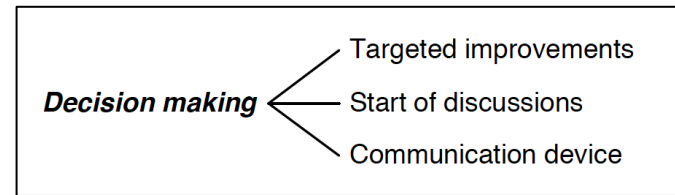


Resulting Coding System



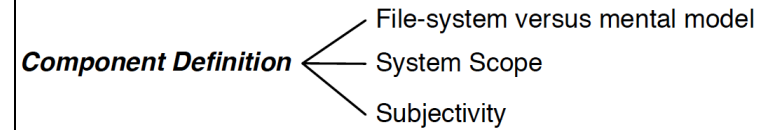
Michaela Greiler, Arie van Deursen, Margaret-Anne D. Storey: Test confessions: A study of testing practices for plug-in systems. ICSE 2012: 244-253

Motivating Refactorings



- Two substantial refactorings mentioned:
 1. Code with semi-deprecated part
 2. Code with wrong top-level decomposition.
- *Developers were aware of need for refactoring. With metrics, they could:*
 - *Explain need to stakeholders*
 - *Explain progress made to stakeholders*

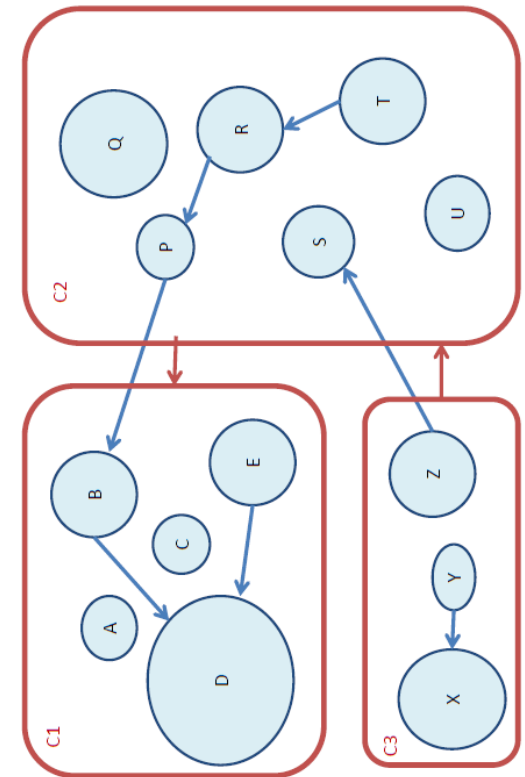
What is a Component?



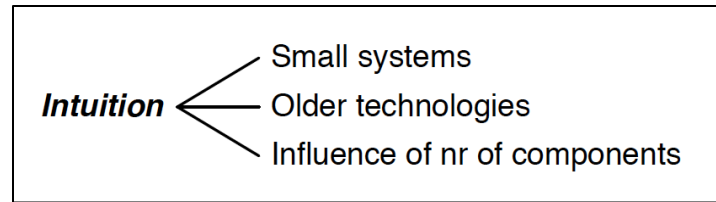
Different “architectures” exist:

1. In the minds of the developers
2. As-is on the file system
3. As used to compute the metrics

- Easiest if 1=2=3
- Regard as different *views*
- Different view per developer?

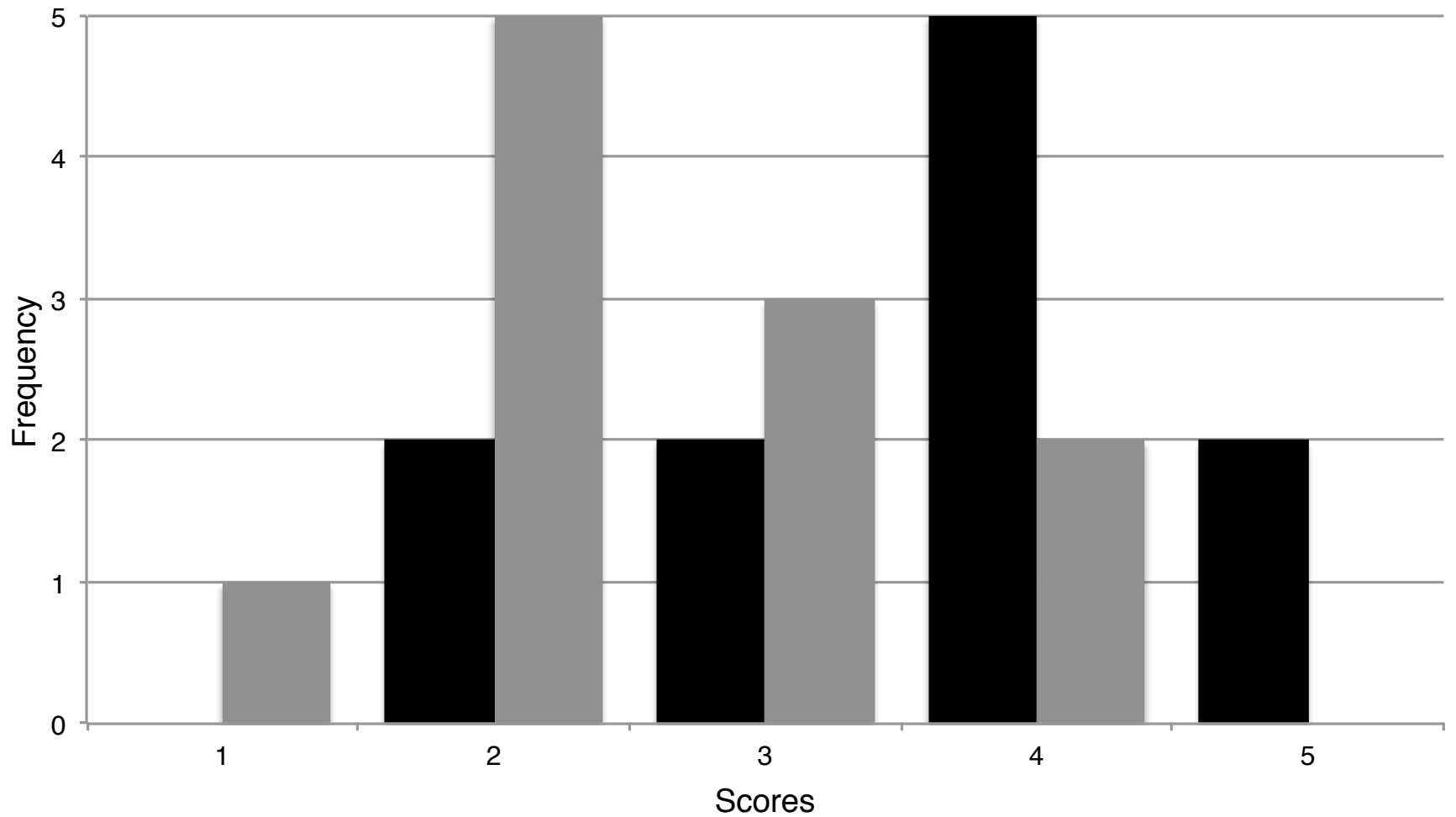


Concerns



- Do size or age affect information hiding?
- No components in Pascal, Cobol, ...
 - Naming conventions, folders, mental, ...
 - Pick best fitting mental view
- # top level components independent of size
 - Metric distribution also not size dependent

Not Easy-to-Use. But Useful.



Dependency Profiles: Conclusions

Lessons Learned

Need for

- Strict component definition guidelines
- Body of knowledge
 - Value patterns
 - With recommendations
 - Effort estimation
- Improved dependency resolution

Threats to Validity

- High realism
- Data confidential
- Range of different systems and technologies

Wanted: replication in open source (Java / Sonar) context

A Summary in Seven Slides



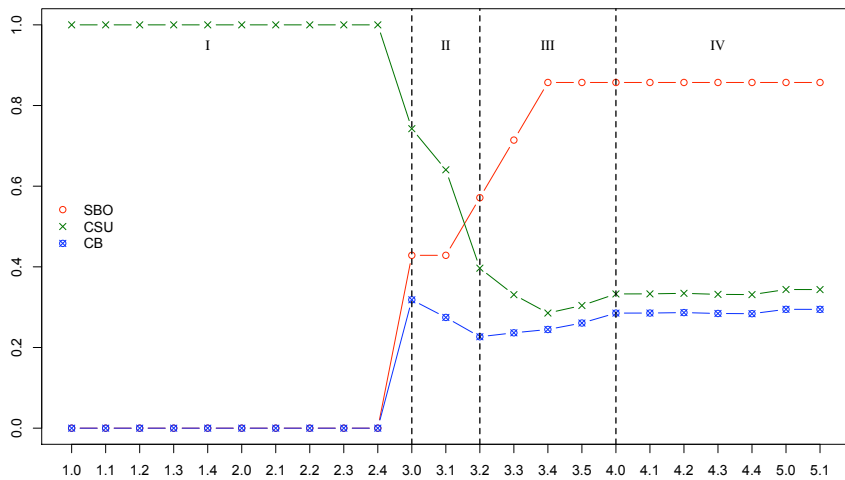
Accountability and Explainability

- Accountability in software architecture?
 - Not very popular
- Stakeholders are entitled to an explanation
- Metrics are a necessary ingredient

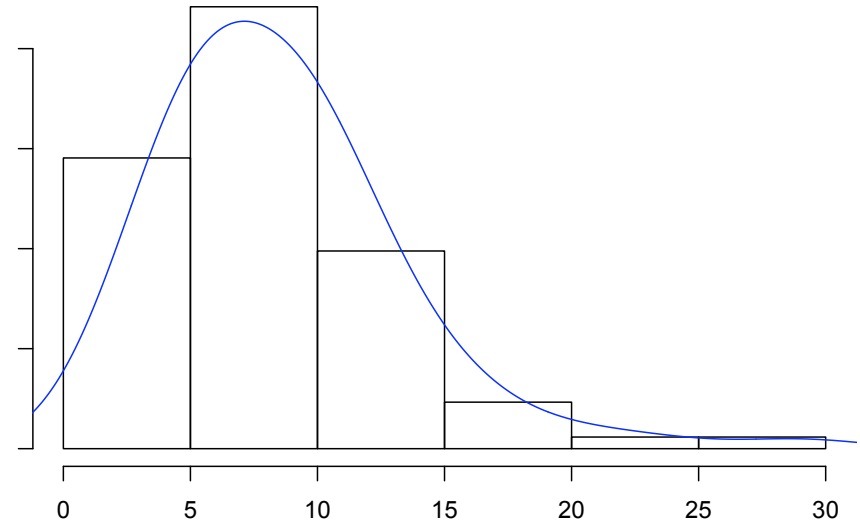


Metrics Need Context

Temporal / Trend



Peers / Norms

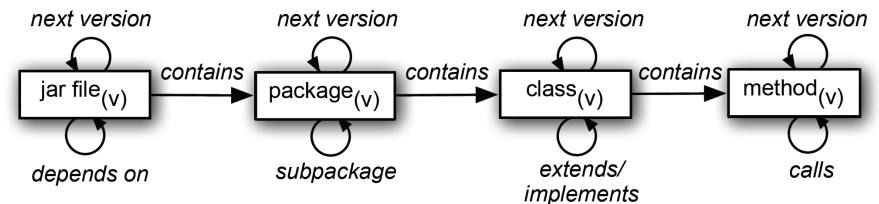
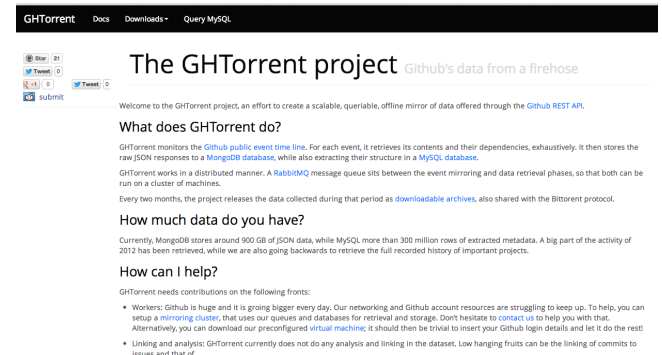


Metrics Research Needs Datasets

Two recent Delft data sets:

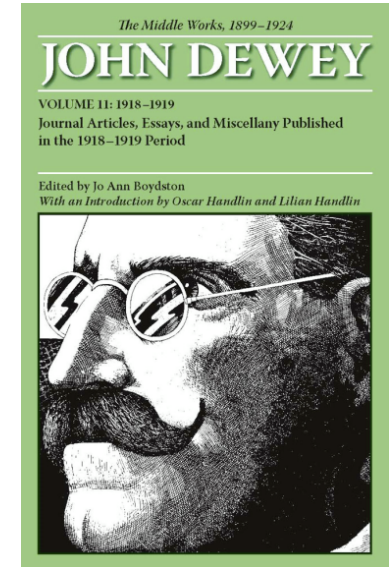
- Github Torrent:
 - Years of github history in relational database.
 - Georgios Gousios
- Maven Dependency Dataset
 - Versioned call-level dependencies in full Maven Central.
 - Steven Raemaekers

ghtorrent.org



Metrics Research needs Qualitative Methods

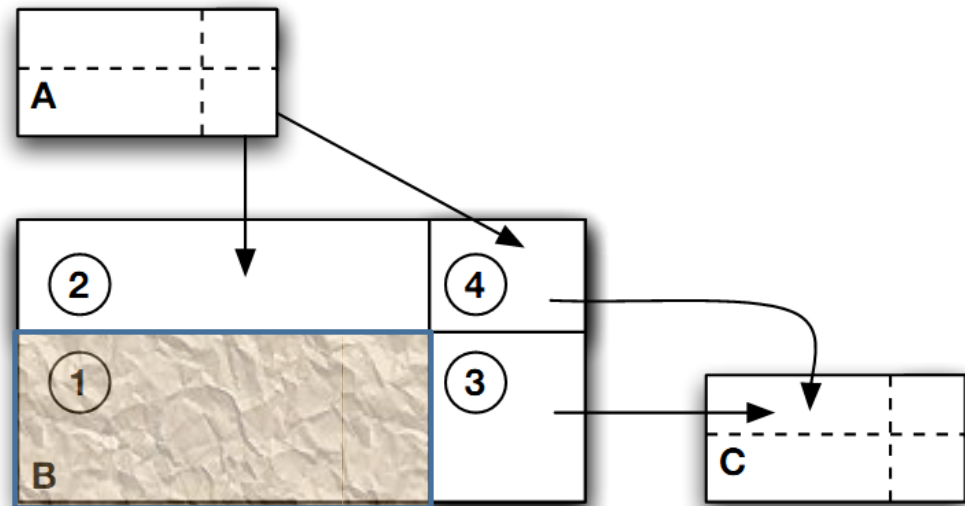
- Evaluate based upon the *possibilities of action*
- Calls for rigorous studies capturing reality in *rich narratives*
- Case studies, interviews, surveys, ethnography, grounded theory, ...



Encapsulation Can be Measured

Module types:

1. Internal
2. Inbound
3. Outbound
4. Transit



And doing so, leads to meaningful discussions.

Should we be Afraid of Change?

Metrics for Software Evolvability



Arie van Deursen, Delft University of Technology
Joint work with Eric Bouwers & Joost Visser (SIG)
@avandeursen