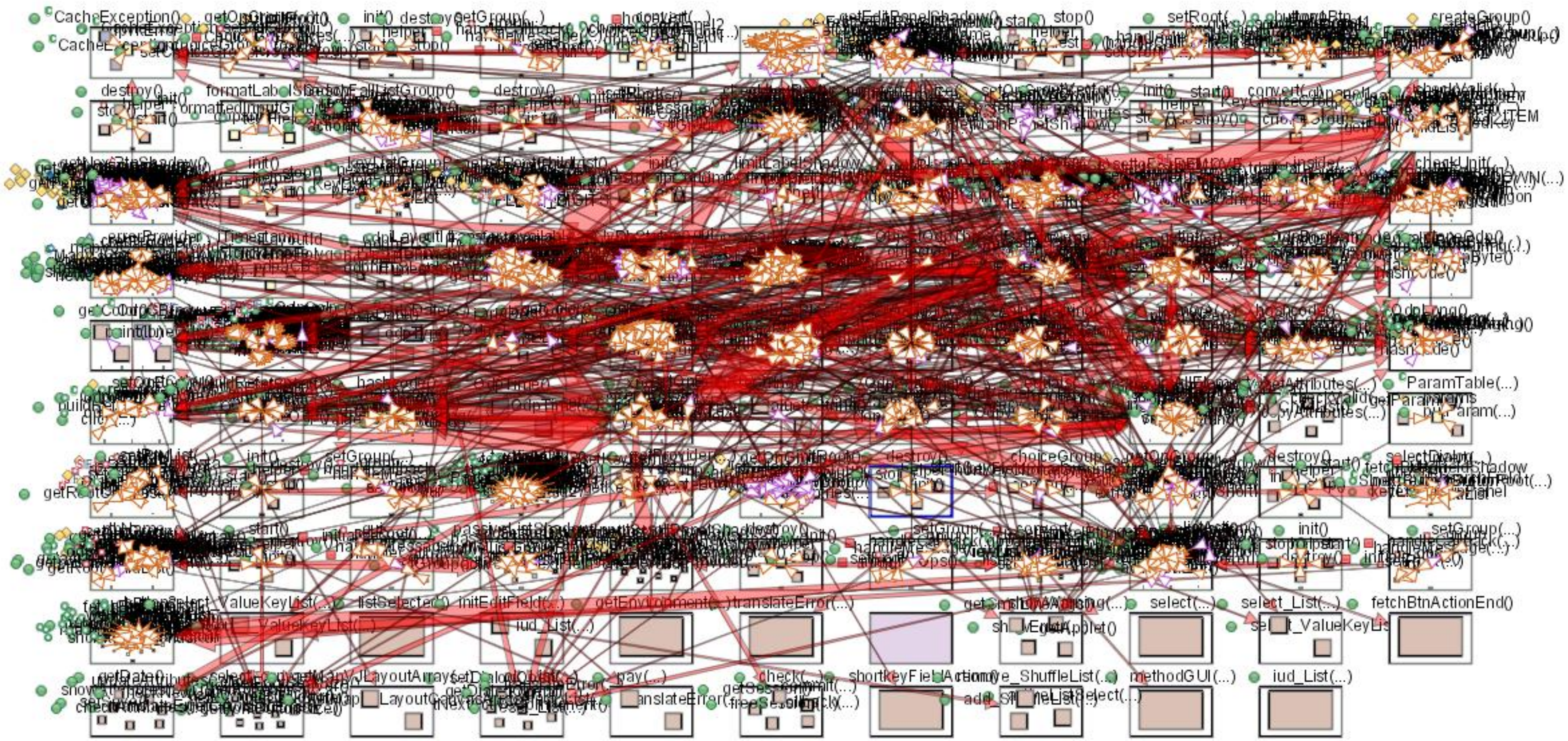
# ReUse: Challenges and Business Success

**Harald Gall**
Department of Informatics
software evolution and architecture lab
seal.ifi.uzh.ch/gall

Universität Zürich

s.e.a.l. software evolution & architecture lab

# Objects and reuse …




… some practical observations

# Outline

- Reuse Challenges
- Reuse Technologies
  - software analysis & visualization
  - product lines, feature engineering & variability
- Economics of reuse
  - cost/benefit relation
  - cost estimation
- Case Studies & Empirical Investigations
- Business Success
- Conclusions

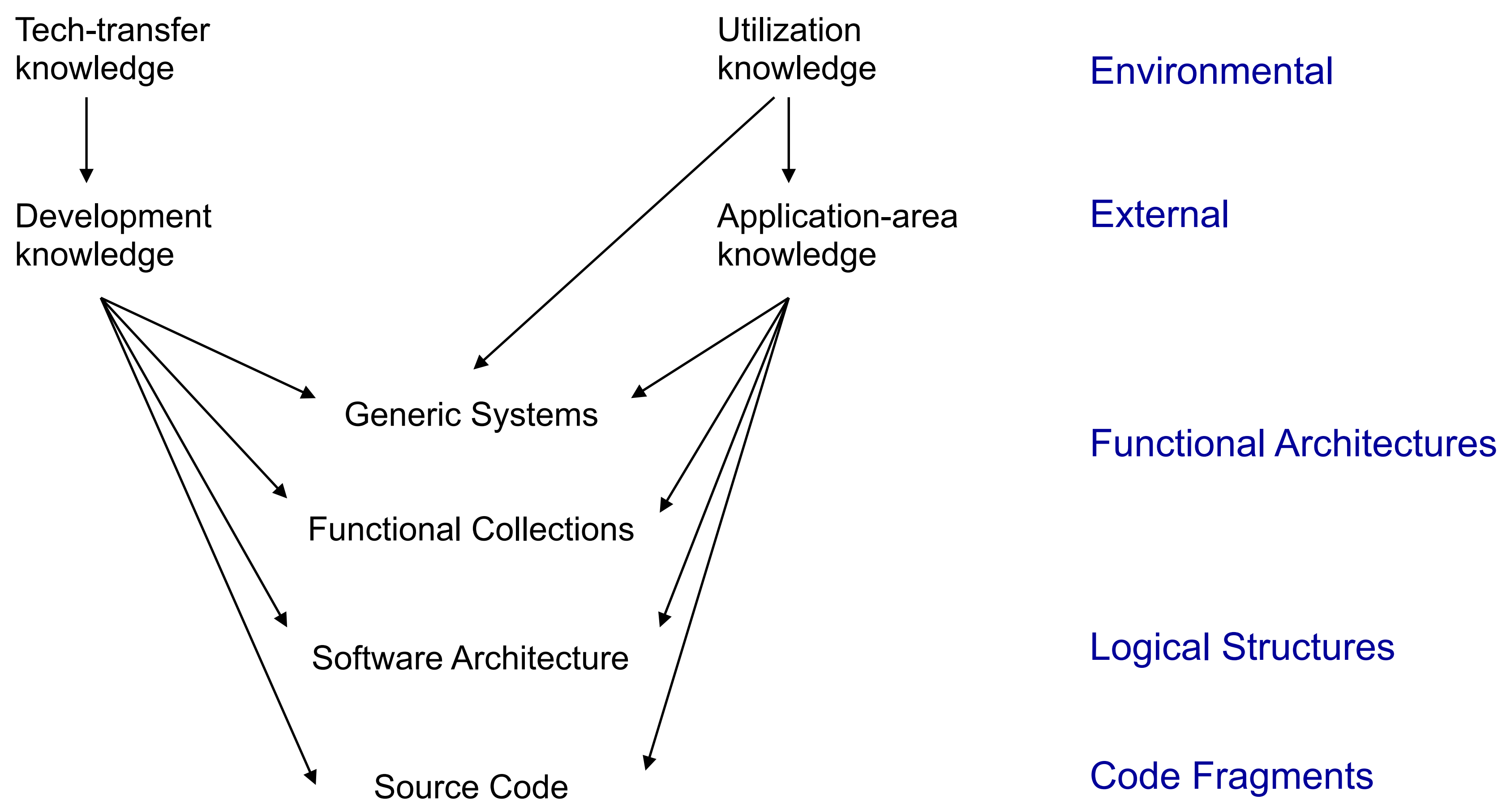
# I. REUSE CHALLENGES

# Software Reuse

- Software Reuse (Mili et al., 2002)

  “Software reuse is the process whereby an organization defines a set of systematic operating procedures to specify, produce, classify, retrieve, and adapt software artifacts for the purpose of using them in its development activities.”

# Reusable Software Engineering (Freeman 1983)




→ B cannot be realized without A

# Challenges of Software Reuse

- Organizational aspects
  - Operational and technological infrastructure
  - Reuse introduction
- Technical aspects
  - Domain engineering
  - Component engineering
  - Application engineering
- Economical aspects
  - Reuse metrics
  - Reuse cost estimation
- Legal aspects
  - Copyright
  - Warranty
  - Open Source

# Challenge to the benefit (1)

| | |
|---|---|
| Increased dependability | Reused software, that has been tried and tested in working systems, should be more dependable than new software. The initial use of the software reveals any design and implementation faults. These are then fixed, thus reducing the number of failures when the software is reused. |
| Reduced process risk | If software exists, there is less uncertainty in the costs of reusing that software than in the costs of development. This is an important factor for project management as it reduces the margin of error in project cost estimation. This is particularly true when relatively large software components such as sub-systems are reused. |
| Effective use of specialists | Instead of application specialists doing the same work on different projects, these specialists can develop reusable software that encapsulate their knowledge. |

(Sommerville, 2010)

# Challenge to the benefit (2)

| | |
|---|---|
| Standards compliance | Some standards, such as user interface standards, can be implemented as a set of standard reusable components. For example, if menus in a user interfaces are implemented using reusable components, all applications present the same menu formats to users. The use of standard user interfaces improves dependability as users are less likely to make mistakes when presented with a familiar interface. |
| Accelerated development | Bringing a system to market as early as possible is o ften more important than overall development costs. Reusing software can speed up system production because both development and validation time should be reduced. |

# Reuse problems (1)

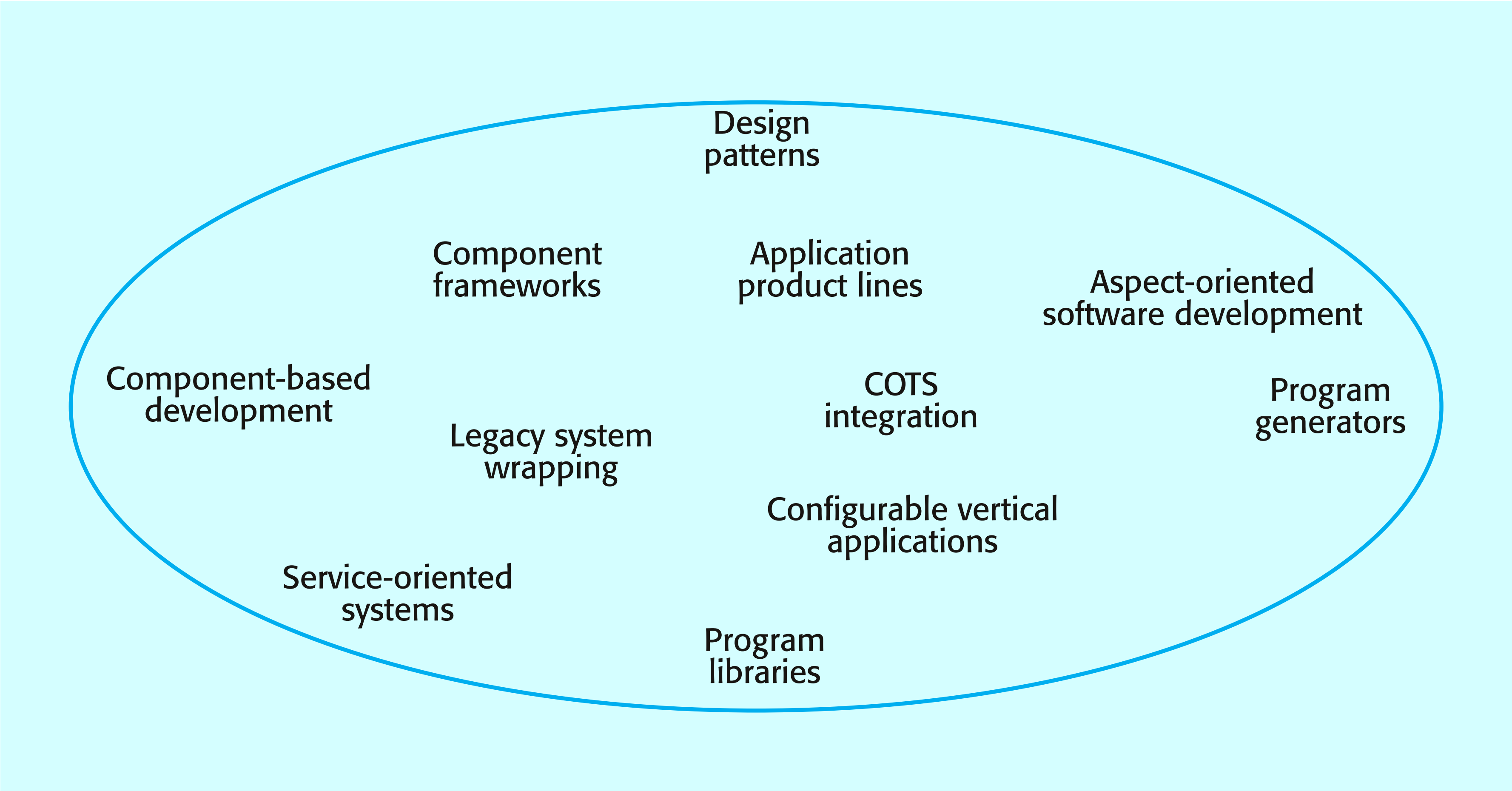
| | |
|---|---|
| Increased maintenance costs | If the source code of a reused software system or component is n ot available then maintenance costs may be increased as the reused elements of the system may become increasingly incompatible with system changes. |
| Lack of tool support | CASE toolsets may not support development with reuse. It may be difficult or impossible to integrate these tools with a component library system. The software process assumed by these tools may not take reuse into account. |
| Not-invented-here syndrome | Some software engineers sometimes prefer to re-write components as they believe that they can improve on the reusable component. This is partly to do with trust and partly to do with the fact that writing original software is s een as more challenging than reusing other people's software. |

# Reuse problems (2)

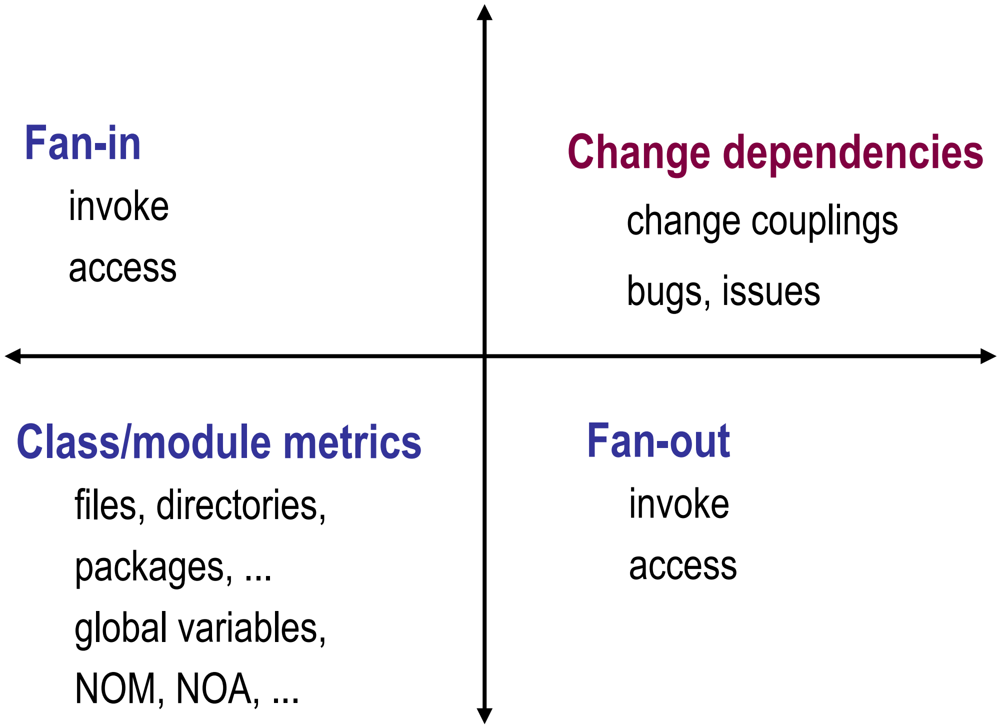
| | |
|---|---|
| Creating and maintaining a component library | Populating a reusable component library and ensuring the software developers can use this library can be expensive. Our current techniques for classifying, cataloguing and retrieving software components are immature. |
| Finding, understanding and adapting reusable components | Software components have to be discovered in a library, understood and, sometimes, adapted to work in a n ew environment. Engineers must be reasonably confident of finding a component in the library before they will make routinely include a component search as part of their normal development process. |

# II. REUSE TECHNOLOGIES

# The reuse landscape

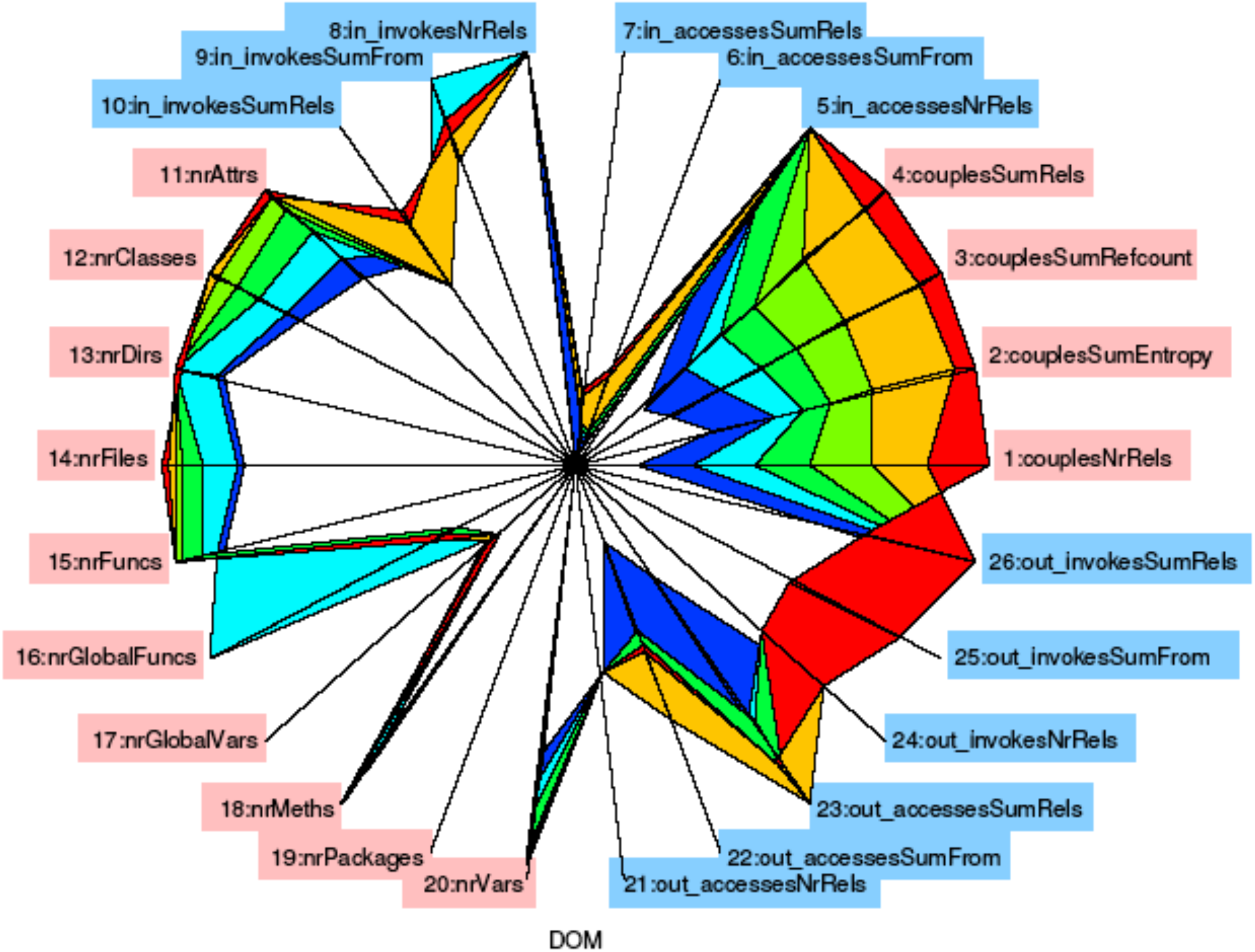


# Software Evolution Metrics

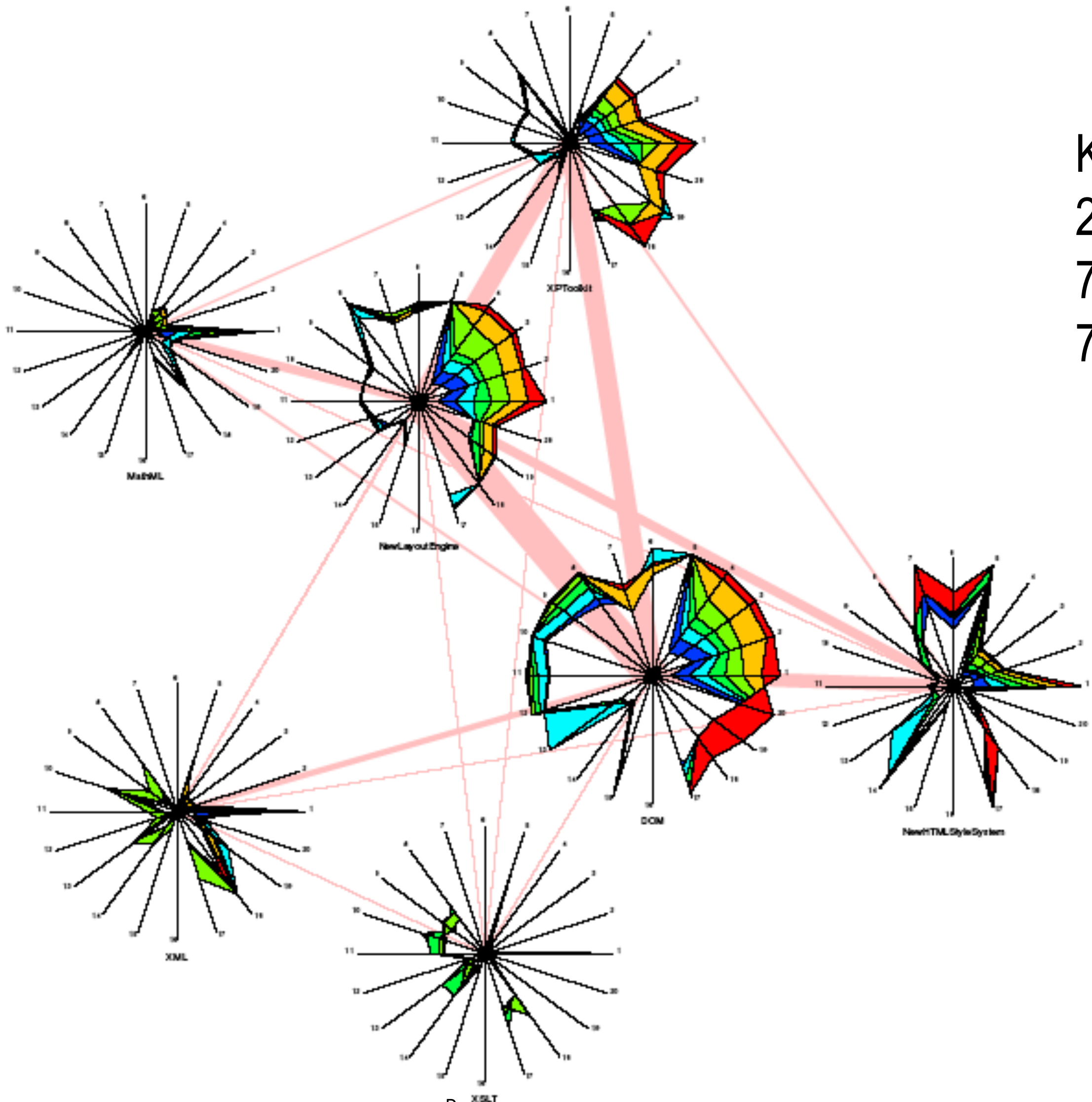



Martin Pinzger, Harald C. Gall, Michael Fischer, and Michele Lanza, *Visualizing Multiple Evolution Metrics*
In Proceedings of the ACM Symposium on Software Visualization, 2005.

# Mozilla Module DOM: 0.92 -> 1.7



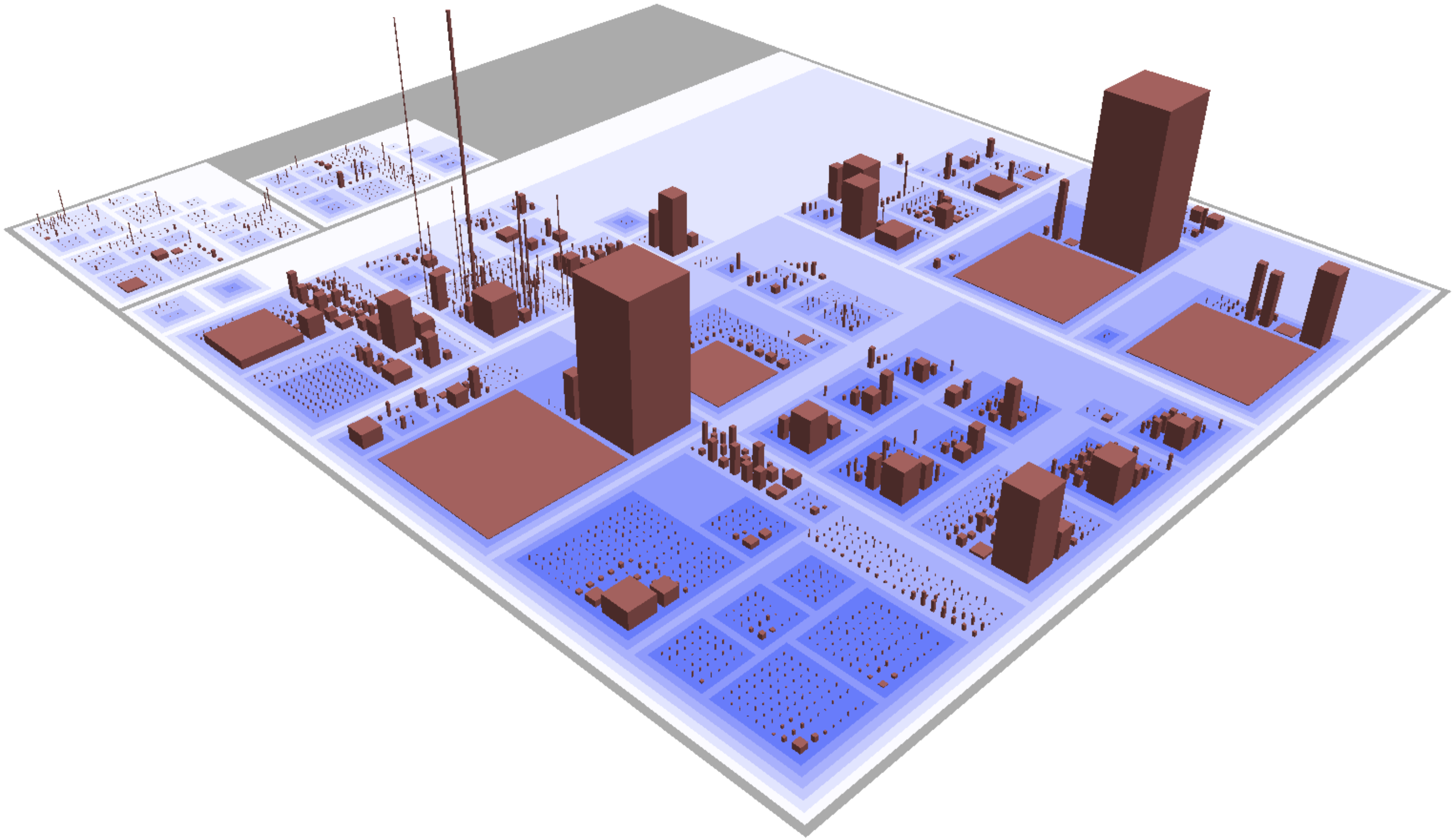
# Mozilla: Change Dependencies




Reuse

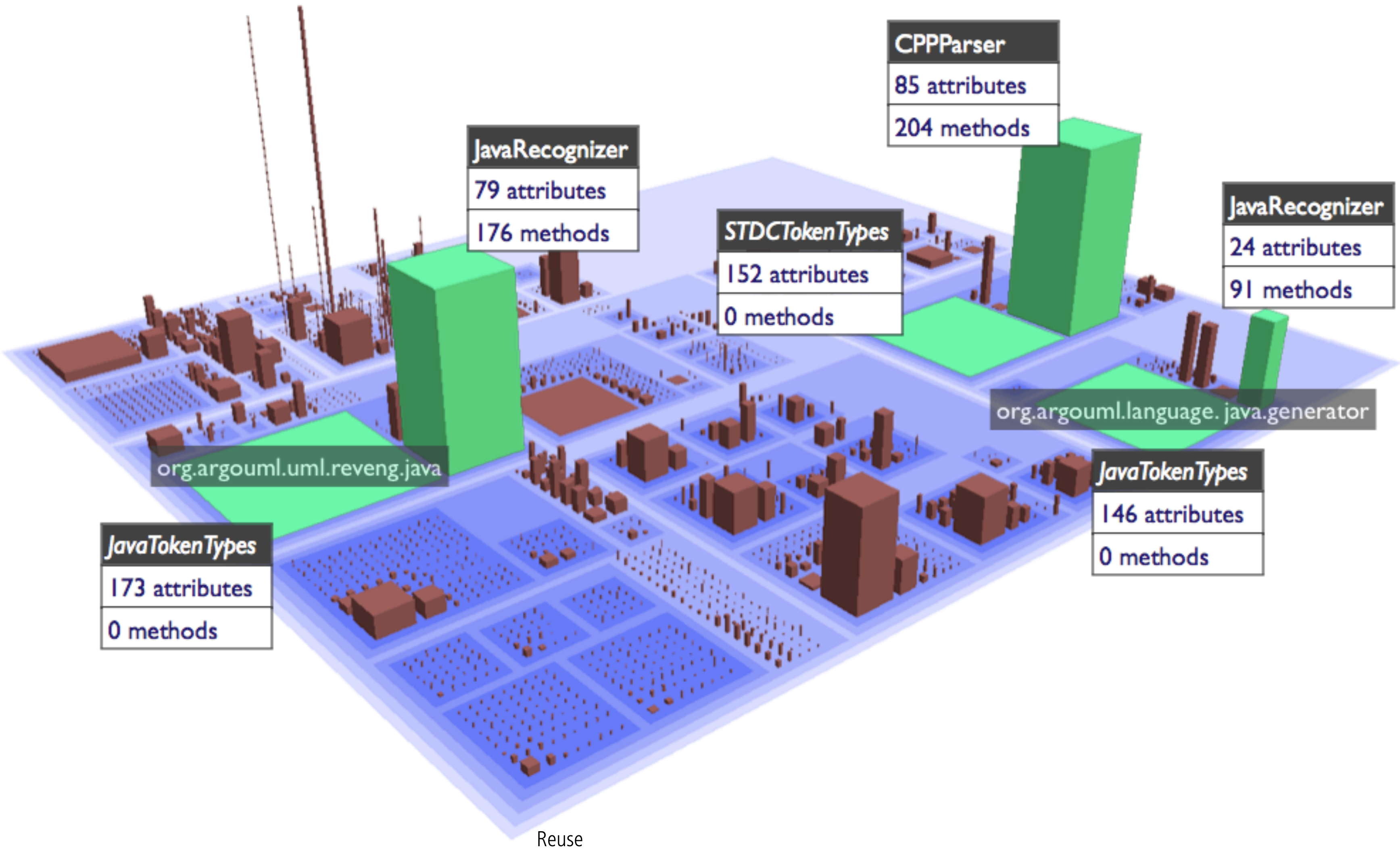
Kiviat graph:
26 metrics
7 Mozilla modules
7 subsequent releases

# Software as City



Richard Wettel, Michele Lanza. Visualizing Software Systems as Cities. In VISSOFT 2007

# Buildings of ArgoUML

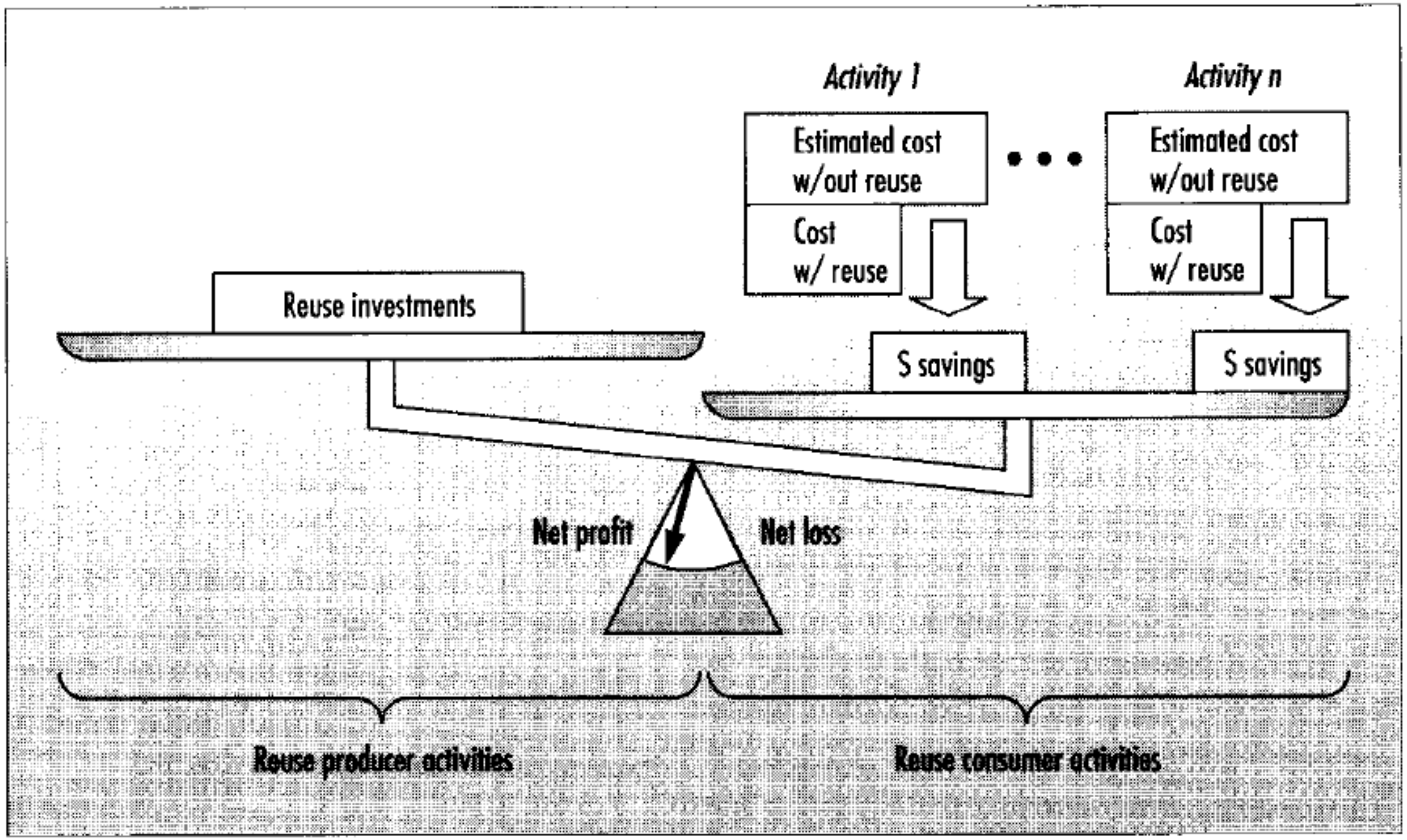


# III. REUSE ECONOMICS

# Reuse investment

- Reuse investment cost
  - cost of producer to provide components for reuse
- Component generality
  - variations of a component in relation to the reuse technology
- Cost of reuse
  - cost of reuser for finding, adapting, integrating, and testing of a reusable component

# Reuse investment relation



# Reuse cost estimation (1)

- $C_{no\text{-}reuse}$ = development cost without reuse
- Reuse Level, $R = \frac{\text{total size of reused components}}{\text{size of application}}$
- $F_{use}$ = relative cost for the reuse of a component
  - typically 0.1 - 0.25 of development cost
- $C_{part\text{-}with\text{-}reuse} = C_{no\text{-}reuse} * (R * F_{use})$
- $C_{part\text{-}with\text{-}no\text{-}reuse} = C_{no\text{-}reuse} * (1 - R)$

- $C_{\mathbf{with\text{-}reuse}} = C_{part\text{-}with\text{-}reuse} + C_{part\text{-}with\text{-}no\text{-}reuse}$
- $C_{\mathbf{with\text{-}reuse}} = C_{no\text{-}reuse} * (R * F_{use} + (1 - R))$

# Reuse cost estimation (2)

- Example: $R = 50\%$, $F_{use} = 0.2$
  - cost for developing with reuse = 60% of cost for developing without reuse
- $C_{saved}$

$$C_{saved} = C_{no\text{-}reuse} - C_{with\text{-}reuse}$$
$$= C_{no\text{-}reuse} * (1 - (R * F_{use} + (1 - R)))$$
$$= C_{no\text{-}reuse} * R * (1 - F_{use})$$

- $ROI_{saved}$

$$ROI_{saved} = \frac{C_{saved}}{C_{no\text{-}reuse}}$$
$$= R * (1 - F_{use})$$

# Reuse cost estimation (3)

- $F_{create}$ = relative cost for the creation and management of a reusable component system
- $C_{component\text{-}systems}$ = cost for developing enough components for R percent
- $F_{create} >> F_{use}$ $1 <= F_{create} <= 2.5$
- $C_{family\text{-}saved} = n * C_{saved} - C_{component\text{-}system}$
  $= C_{no\text{-}reuse} * (n * R * (1 - F_{use}) - R * F_{create})$

# Reuse cost estimation (4)

$$ROI = \frac{C_{family\text{-}saved}}{C_{component\text{-}systems}} = \frac{n * R * (1 - F_{use}) - R * F_{create}}{R * F_{create}}$$

$$= \frac{n * (1 - F_{use}) - F_{create}}{F_{create}}$$

Example: $F_{use} = 0.2$ and $F_{create} = 1.5$

$$ROI = \frac{n * 0.8 - 1.5}{1.5}$$

Break-even mit $n > 2$

# COPLIMO – Software Product Line Life Cycle Cost Estimation

(Boehm et al., 2004)

**Table 8. Relative Development Effort Without and With Product Line Reuse**

| # Products N / Effort (PM) | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| No Reuse | 294 | 588 | 882 | 1176 | 1470 |
| Product Line | 444 | 593 | 742 | 891 | 1040 |
| Product Line Savings | -150 | -5 | 140 | 285 | 430 |
| ROI=PLS(N)/\|PLS(1) | -1.0 | -.03 | +.93 | +1.9 | +2.9 |

| | |
|---|---|
| Product-specific software (PFRAC): | 0.4 |
| Black-box plug-and-play reuse (RFRAC): | 0.3 |
| Reuse with modifications (AFRAC): | 0.3 |
| Assessment and assimilation factor (AA): | 2 |
| Software understanding increment (SU): | 10 |
| Unfamiliarity factor (UNFM): | 0.5 |
| % design modified (DM): | 15% |
| % code modified (CM): | 30% |
| % integration redone (IM): | 40% |

# Relative Cost of Writing for Reuse

- RCWR is the added cost of writing software to be most cost-effectively reused across a product line family of applications, relative to the cost of writing a standalone application.

- $C_{RCWR}$ = LaborRate * $COPLIMO_{RCWR}$ + $SoftwareQualityCost_{RCWR}$

- $C_{RCWR}$ = LaborRate * [COCOMO baseline (initialSoftwareSize) * EffortAdjustment for RCWR] + [ CostPerDefect * (1- TestingEffectiveness) * (COQUALMO(initialSoftwareSize, $EM_{PL}$)],

*where $EM_{PL}$ is the Effort Multiplier of the COCOMO II cost drivers for the product line development and COCOMO baseline is calculated as 2.94 * (software size$^{1.0997}$ * PI(EM)*

(Boehm et al., 2006)

# Relative Cost for Reuse

- RCR is the cost of reusing the software in a new application with the same product line family, relative to developing newly built software for the application.

- $C_{RCR}$ = LaborRate * $COPLIMO_{RCR}$ + $SoftwareQualityCost_{RCR}$

- $C_{RCR}$ = LaborRate * [COCOMO baseline (softwareSizeForReuse)] + [CostPerDefect * (1 – TestingEffectiveness) * COQUALMO(softwareSizeForReuse, $EM_{PL}$)]

(Boehm et al., 2006)

# Estimated quality-based SPL cost

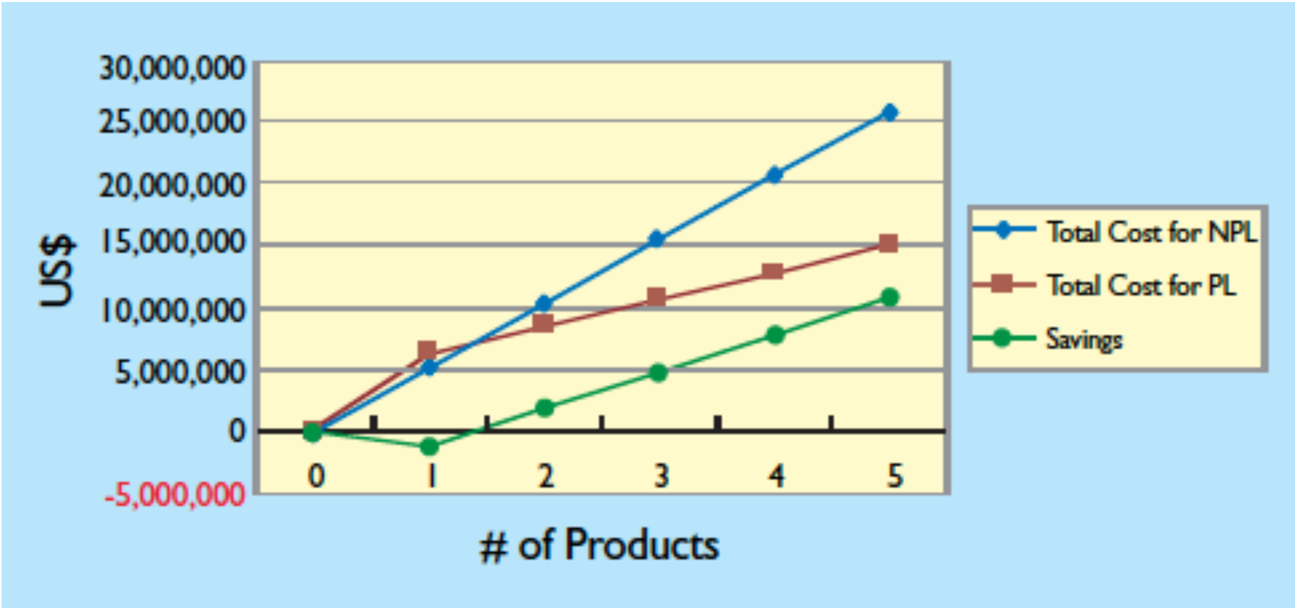
- $C_{PL}(N) = C_{RCWR} + (N-1) * C_{RCR}$
  *where N is the number of products to be developed in SPL*

| Parameters | Values | Parameters | Values |
|---|---|---|---|
| Initial Software Size | 100 KSLOC | Software size for reuse | 50.11 KSLOC |
| LaborRate | $8,000 / MM | Effort Adjustment for RCWR[1] | 1.469362 |
| $EM_{NPL}$ (all cost drivers are Nominal) | 1.0 | Testing Effectiveness (TE) | 0.9 |
| $EM_{PL}$[2] | 1.78227 | Cost per Defect (CD) | $10,000 |

$$C_{PL}(N) = \$6'333 + (N - 1) * \$2'174$$

(Boehm et al., 2006)

# Saving of NPL vs. PL



# IV. CASE STUDIES

# A. HP case study

| Case study | Period | Non-comment source statements | Programming language | Development OS | Target OS |
|---|---|---|---|---|---|
| HP's Manufacturing Productivity Section | 1983-1994+ | 55 KNCSS (685 reusable workproducts) | Pascal, SPL | MPEXL für HP3000 | MPEXL |
| HP's San Diego Graphics Division | 1987 - 1994+ | 20 KNCSS | C | HPUX | PSOS |

aus W.C. Lim, IEEE Software, Sept. 1994

# Reuse program economic profiles

| Organization | Manufacturing Productivity | San Diego Technical Graphics |
|---|---|---|
| Time horizon | 1983-1992<br>(10 years) | 1987-1994<br>(8 years) |
| Start-up resources required | 26 engineering months<br>(start-up costs for 6 products)<br><br>USD 0.3 million | 107 engineering months<br>(3 engineers for 3 years)<br><br>USD 0.3 million |
| Ongoing resources required | 54 engineering months<br>(1/2 engineer for 9 years)<br><br>USD 0.3 million | 99 engineering months<br>(1-3 engineers for 5 years)<br><br>USD 0.7 million |
| Gross cost | 80 engineering months<br><br>USD 1 million | 206 engineering months<br><br>USD 2.6 million |
| Gross savings | 328 engineering months<br><br>USD 4.1 million | 446 engineering months<br><br>USD 5.6 million |
| **Return on Investment (savings/cost)** | **410%** | **216%** |
| Net present value | 125 engineering months<br>USD 1.6 million | 75 engineering months<br>USD 0.9 million |
| **Break-even year (recoup start-up)** | **Second year** | **Sixth year** |

aus W.C. Lim, IEEE Software, Sept. 1994

# Quality, productivity, time-to-Market

| Organization | Manufacturing Productivity | San Diego Technical Graphics |
|---|---|---|
| Quality | 51% defect density reduction compared to new code | 24% defect density reduction |
| Reused code | 38% | 31% |
| Productivity | 57% increase over development from scratch | 40% increase |
| Time-to-market | n.v. | 42% reduction |

aus W.C. Lim, IEEE Software, Sept. 1994

# Reuse cost

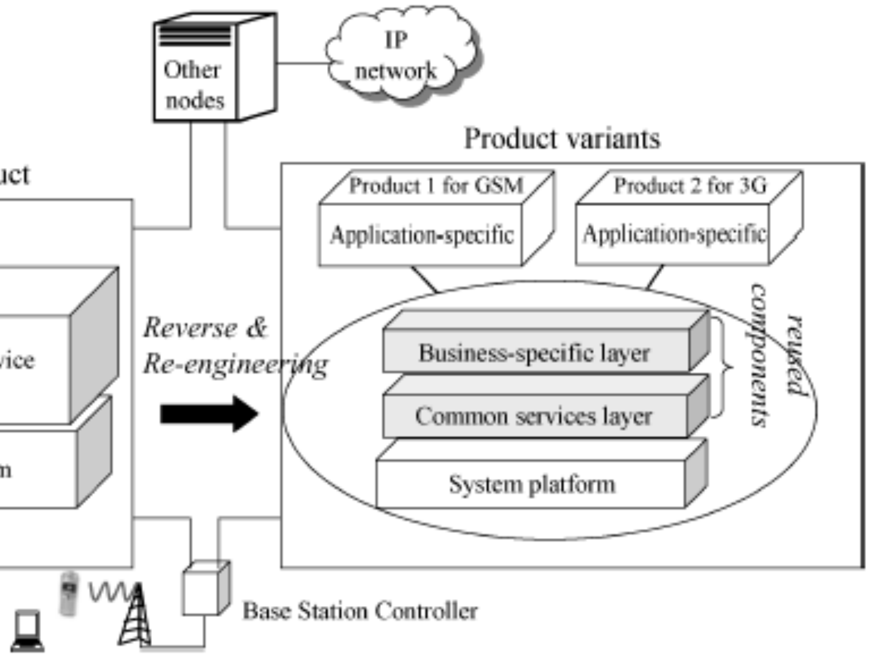
| Domain | Air-traffic-control System | Menu- und Forms-Mgmt System | Graphics Firmware |
|---|---|---|---|
| Relative cost to create reusable code | 200 % | 120 - 480% | 111% |
| Relative cost to reuse | 10 - 20% | 10 - 63% | 19% |

aus W.C. Lim, IEEE Software, Sept. 1994

# B. Ericsson study (2008)

(Mohagheghi & Conradi, 2008)

- 3y software reuse in 2 large telecom products (Norway and Sweden)
- reused components were developed in-house and shared in a product-family approach
- reuse as risk mitigation since development moved to Sweden
- quantitative data mined and qualitative observations



# Ericsson study, continued

- Component-based architecture (CORBA)
- Components programmed in Erlang, C, and some Java (GUI)
- Data analyzed:
  - Trouble Reports: failures observed by testers or users
  - Change Requests: changes to requirements after baseline
  - KLOC and modified KLOC between releases
  - Person Hours used in system test
  - code modification rate: (m-KLOC/KLOC)*100
  - reuse rate: size of reused code

# Ericsson study, continued

- Quality benefits of large-scale reuse programs
  - significant benefits in terms of lower fault density and
  - less modified code between releases of reused code
  - reuse reduced risks and lead time of second product since it was developed based on a tested platform
  - reuse and standardization of software architecture, processes and skills can help reduce organizational restructuring risks
- Study showed that there is a need to adapt software processes such as RUP for reuse, and define metrics to evaluate corporate/project/software goals

# V. BUSINESS SUCCESS

# Strategies for Software Reuse

(Rothenberger et al., 2003)

- Potential reuse adopters must be able to understand reuse strategy alternatives and their implications
- Organizations must make an informed decision
- The study:
  - survey data from 71 software development groups (of 67 different organizations), 80% working in organizations > 200 employees
  - software engineers, development consultants, project managers, software engineering researchers
  - to empirically analyze dimensions that describe the practices employed in reuse programs
  - classify reuse settings and assess their potential for success

# Reuse archetypes

| Reuse Setting | Organizational Dimensions | | | Development Environment Dimensions | |
|---|---|---|---|---|---|
| | Planning & Improve-ment | Formalized Process | Mgmt. Support | Project Similarity | Common Architecture |
| **Ad-Hoc Reuse with High Reuse Potential** | low | low | low | high | high |
| **Uncoordinated Reuse Attempt with Low Reuse Potential** | low | low | medium | medium | low |
| **Uncoordinated Reuse Attempt with High Reuse Potential** | medium | low | medium | medium | high |
| **Systematic Reuse with Low Management Support** | medium | medium | low | high | medium |
| **Systematic Reuse with High Management Support** | high | high | high | high | high |

(Rothenberger et al., 2003)

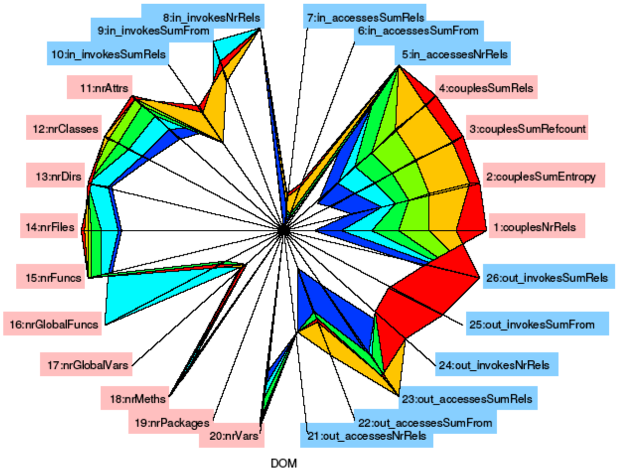
# Software Reuse Strategies: Findings

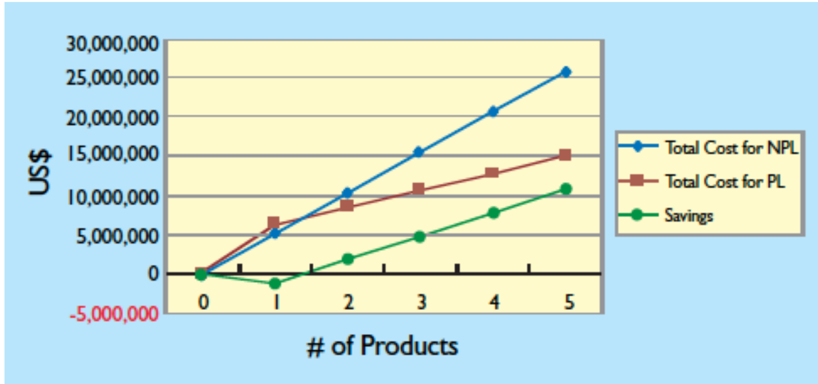
- An organization's reuse success is not dependent on the use of object-oriented techniques. Nevertheless, object technologies may be conducive to reuse, yet the other dimensions ultimately determine reuse success.
- The qualitative analysis yielded additional insights:
  - An improvement of software quality can be achieved without an emphasis on the reuse process
  - An organization will only obtain the full benefit of reuse if a formal reuse program is employed and subject to quality control through formal planning and continuous improvement.

(Rothenberger et al., 2003)

# CONCLUSIONS

# Conclusions



# References


- B. Boehm, A Winsor Brown, R. Madachy, Y. Yang, A Software Product Line Life Cycle Cost Estimation Model, in Proceedings of the 2004 Symposium on Empirical Software Engineering, IEEE, 2004.
- B.H. Peter In, J. Baik, S. Kim, Y. Yang, B. Boehm, Quality-Based Cost Estimation Model for the Product Line Life Cycle, Communications of the ACM, Vol. 49(12), Dec 2006.
- W.C. Lim, Effects of reuse on quality, productivity, and economics, IEEE Software, Vol.11(5), pp.23-30, Sep 1994, doi: 10.1109/52.311048
- D. Lucredio, K. dos Santos Brito, A. Alvaro, V.C. Garcia, E.S. de Almeida, R.P. de Mattos Fortes, S.L.Meira, Software reuse: the Brazilian indudustry scenario, Journal of Systems and Software, 81, Elsevier, 2008.
- H. Mili, A. Mili, S. Yacoub, E. Addy, Reuse Based Software Engineering: Techniques, Organizations, and Measurement, Wiley, 2001.
- P. Mohagheghi, R. Conradi, An Empirical Investigation of Software Reuse Benefits in a Large Telecom Product, ACM Transactions on Software Engineering and Methodology, Vol. 17(3), June 2008.
- P. Mohagheghi, R. Conradi, O.M. Killi, H. Schwarz, An Empirical Study of Software Reuse vs. Defect-Density and Stability, in Proceedings of the 26th International Conference on Software Engineering (ICSE), IEEE, 2004.
- M.A. Rothenberger, K.J. Dooley, U.R. Kulkarni, Strategies for Software Reuse: A Principal Component Analysis of Reuse Practices, Transactions on Software Engineering, Vol. 29(8), Sept. 2003.
- K.C. Desouza, Y. Awazu, A. Tiwana, Four Dynamics for Bringing Use back into Software Reuse, Communications of the ACM, Vol. 49(1), Jan. 2006.
- T. Ravichandran, M.A. Rothenberger, Software Reuse Strategies and Component Markets, Communicatinos of the ACM, Vol. 46(8), Aug. 2003.

# BACKUP

# COCOMO II - Software Understanding

(Boehm et al., 2004)

| | *Very low* | *Low* | *Nominal* | *High* | *Very high* |
|---|---|---|---|---|---|
| ***Structure*** | Very low cohesion, high coupling, spaghetti code | Moderately low cohesion, high coupling | Reasonably well-structured; some weak areas | High cohesion, low coupling | Strong modularity, information-hiding in data and control structures |
| ***Application clarity*** | No match between program and application world views | Some correlation between program and application | Moderate correlation between program and application | Good correlation between program and application | Clear match between program and application world views |
| ***Self-descriptiveness*** | Obscure code; documentation missing, obscure or obsolete | Some code commentary and headers; some useful documentation | Moderate level of code commentary, headers, documentation | Good code commentary and headers; useful documentation; some weak areas | Self-descriptive code; documentation up-to-date, well-organized, with design rationale |
| ***SU increment*** | **50** | **40** | **30** | **20** | **10** |

COCOMO II rating for software understanding

# COCOMO II - Assessment & Assimilation effort

(Boehm et al., 2004)

| Assessment and Assimilation increment | Level of assessment and assimilation effort |
|---|---|
| 0 | None |
| 2 | Basic component search and documentation |
| 4 | Some component test and evaluation |
| 6 | Considerable component test and evaluation |
| 8 | Extensive component test and evaluation |



48