Refactoring
Lecture 6:
Software Engineering

DAT159/H18

Volker Stolz

Supported by the bilateral SIU/CAPES project "Modern Refactoring" 2017/18

Høgskulen på Vestlandet

What do Developers Think?

- "**A Field Study of Refactoring Challenges and Benefits**",
  FSE '12 Proceedings of the ACM SIGSOFT 20th Intl. Symp. on the Foundations of Software Engineering
  http://web.cs.ucla.edu/~miryung/Publications/fse2012-fieldrefactoring.pdf
  Miryung Kim, Thomas Zimmermann, Nachiappan Nagappan
- "**An Empirical Study of Refactoring Challenges and Benefits at Microsoft**"
  IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 40, NO. 7, JULY 2014
  https://ieeexplore.ieee.org/iel7/32/6848876/06802406.pdf

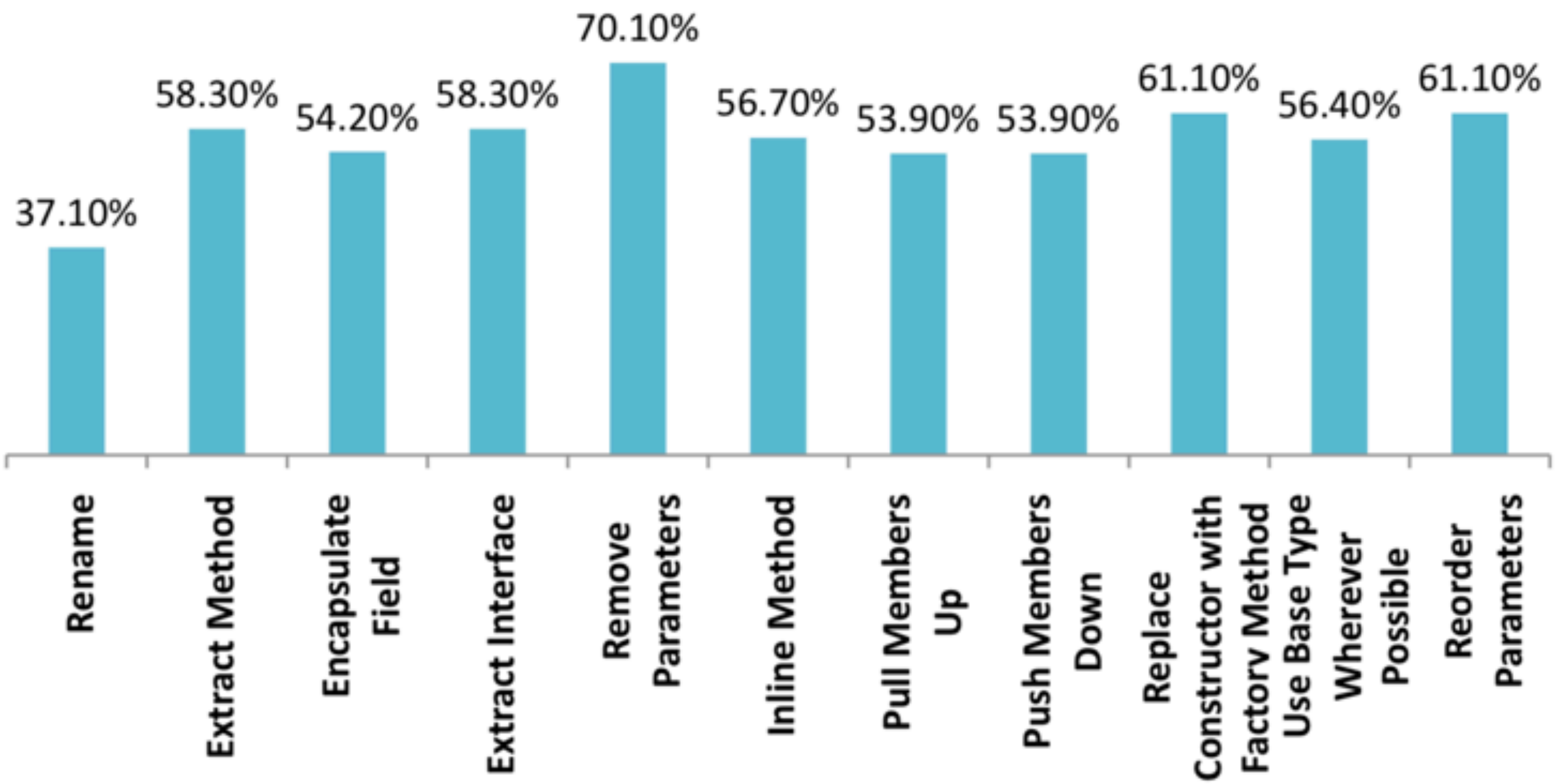
  (accessible from HVL network!)

Refactoring at Microsoft

- Surveyed…
  - 1290 Microsoft engineers;
  - that made changes with the keyword "*refactor**";
  - in Windows Phone, Exchange, Windows, office communication and services (OCS), and Office

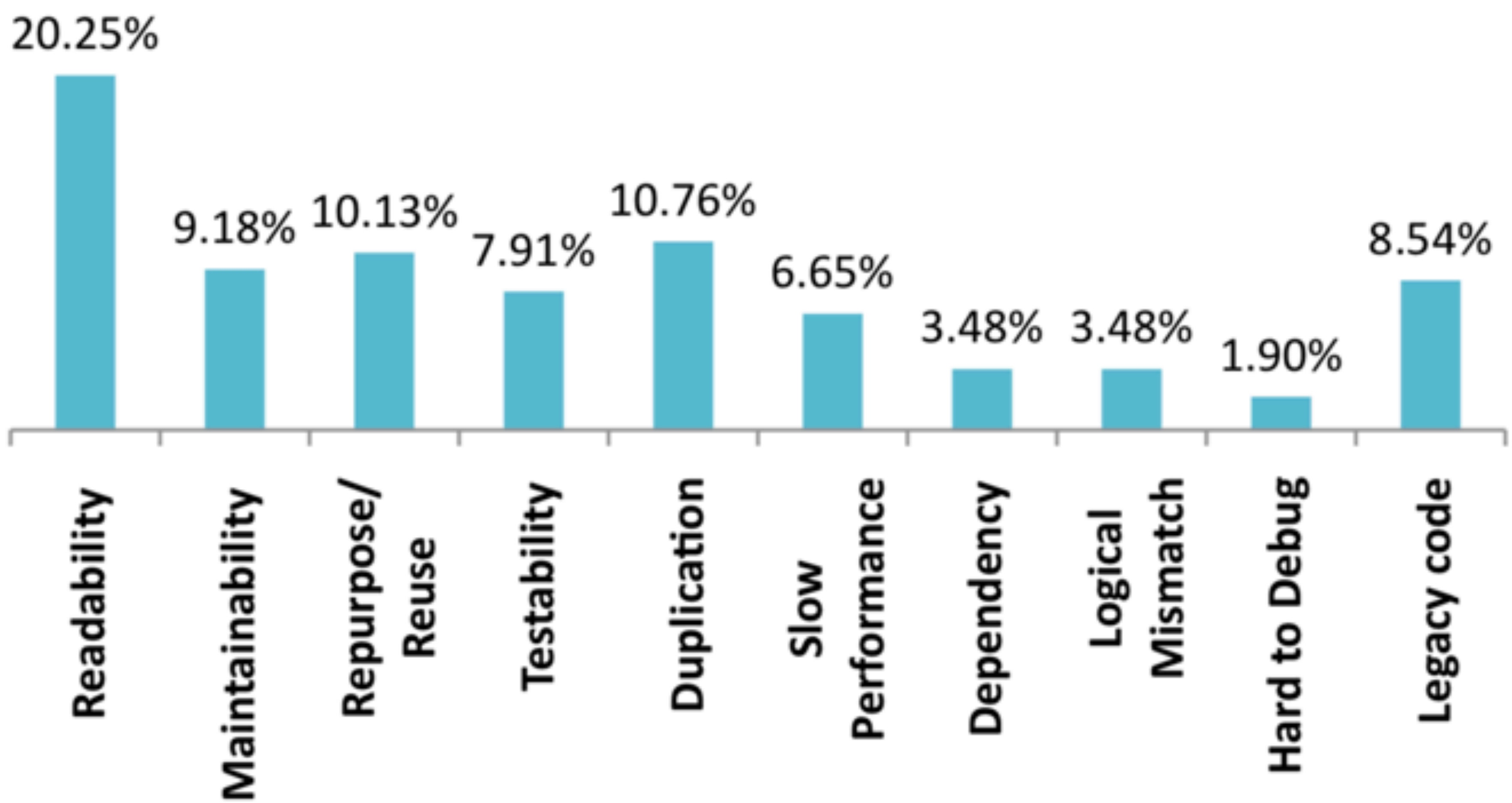
Manual Refactoring

Many developers know about refactorings, but still do them manually:



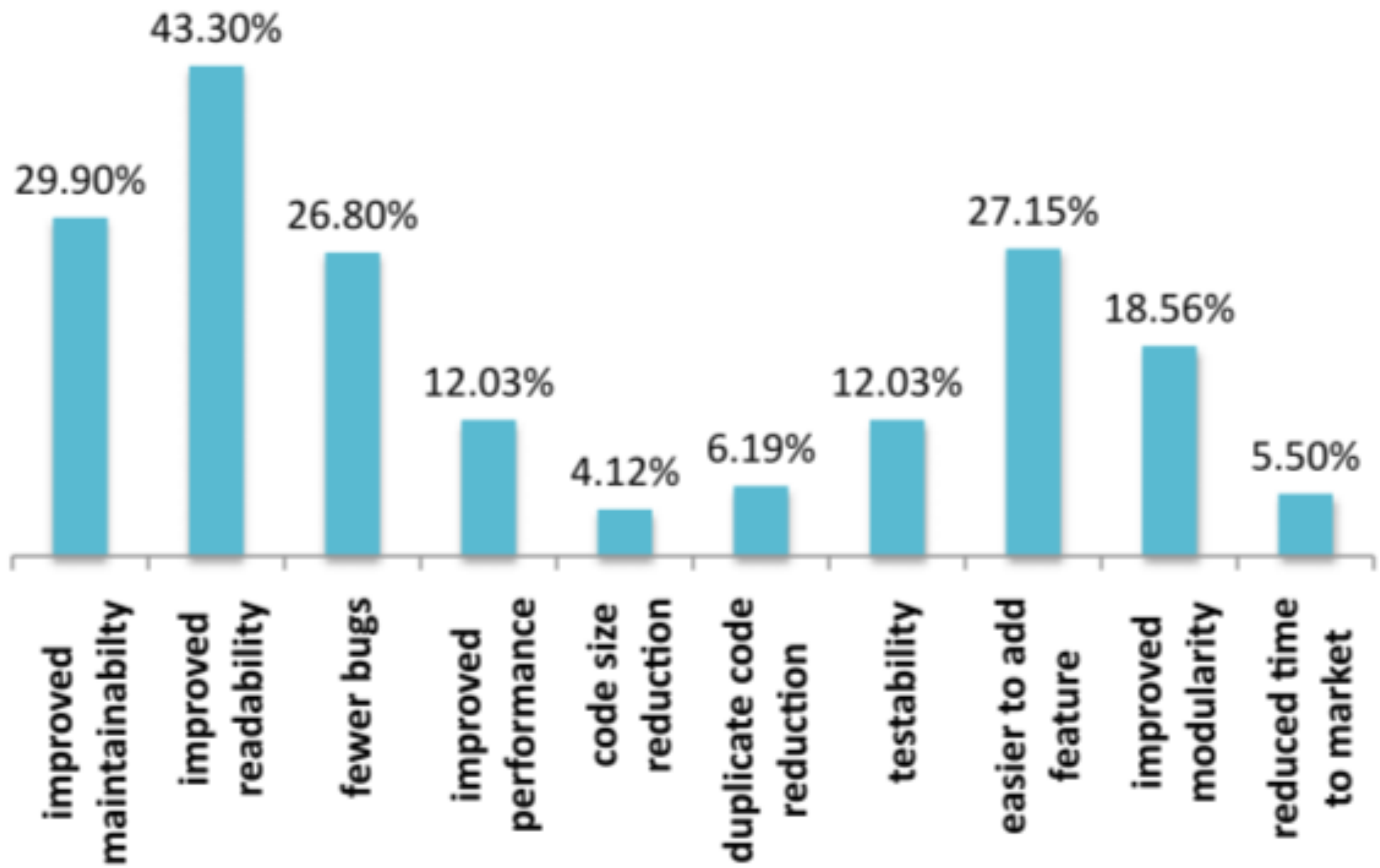
When do developers refactor?

Symptoms that prompt refactoring:



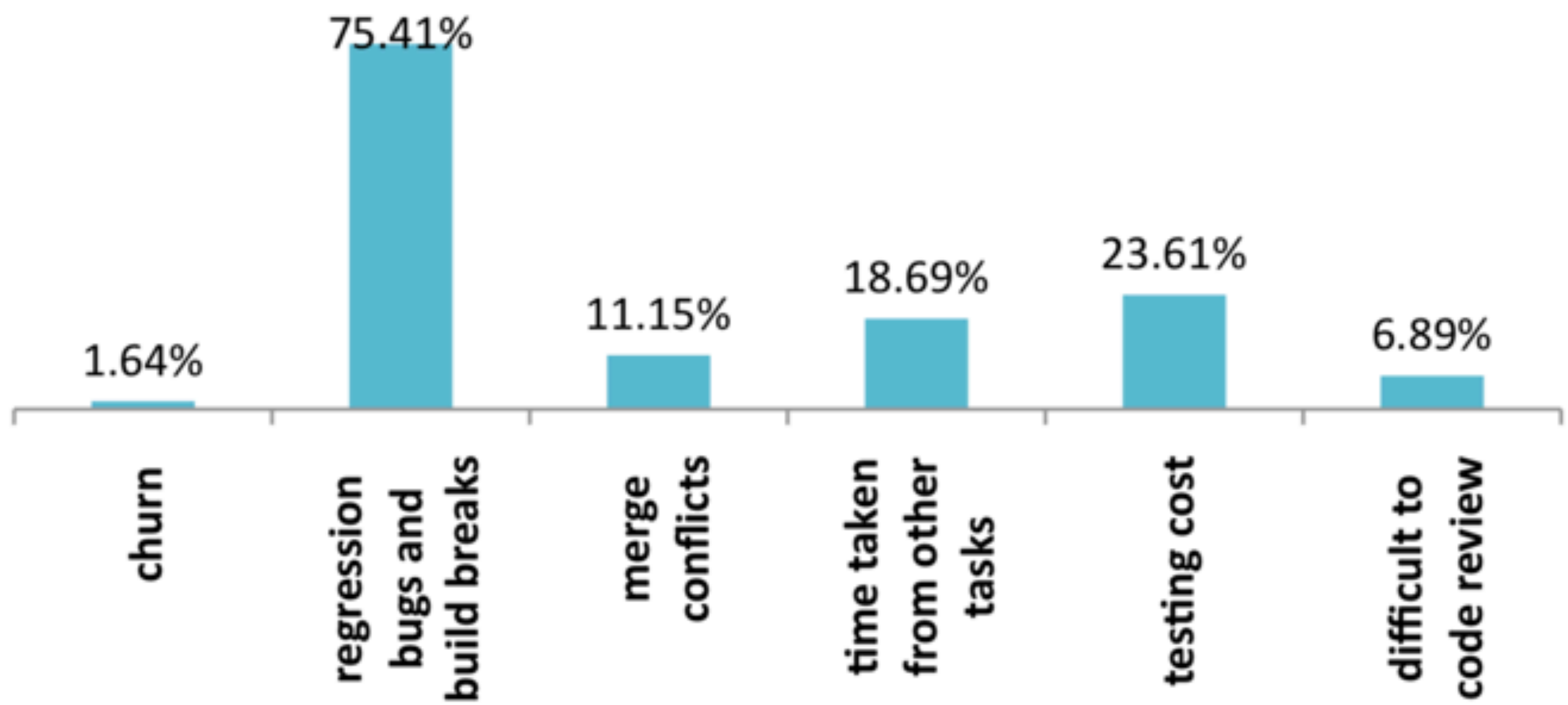
Perceived Benefits

Various types of refactoring benefits that developers experienced:



Risks

Risk factors associated with refactoring:



Refactoring and Bugs

- High correlation between *code churn* and defects in Windows code base in general (Nagappan/Ball, ICSE 2005)
- Weak correlation between *refactoring churn* and defects: refactoring changes are less likely to lead to post release defects than regular changes

Code base	Post Release Defects (Win7)
Top 20% modified	55 %
Top 20% refactored	42 %
Top 40% modified	77,2 %
Top 40% refactored	60,3 %

Hypotheses

Modularity	H1.A: Refactoring was preferentially applied to the modules with a large number of inter-module dependencies.	Confirmed
	H1.B: Preferential refactoring is correlated to changes in the number of inter-module dependencies.	Confirmed
Defect	H2.A: Refactoring was *not* preferentially applied to the modules with a large number of post-release defects.	Confirmed
	H2.B: Preferential refactoring is correlated to reduction in the number of defects.	Rejected
Complexity	H3.A: Refactoring was preferentially made to the modules with high complexity.	Rejected
	H3.B: Preferential refactoring is correlated with reduction in complexity.	Rejected
Size	H4.A: Refactoring was preferentially applied to the modules with large size and preferential refactoring is correlated with size reduction.	Rejected
Churn	H4.B: Refactoring was preferentially applied to the modules where a large number of edits or commits, and preferential refactoring is correlated with the decrease in churn measures.	Rejected
Locality	H4.C: Refactoring was preferentially applied to the modules where logical changes tend to be crosscutting and scattered, and preferential refactoring is correlated with the decrease in the number of crosscutting changes.	Rejected
Developer and Organization	H5.A: Refactoring was preferentially applied to the modules touched by a large number of developers.	Rejected
	H5.B: Refactoring was preferentially applied to the modules that are not cohesive in terms of organizational contributions.	Confirmed
	H5.C: Refactoring was preferentially applied to the modules that are diffused in terms of organizations and developer contribution.	Confirmed
Test Coverage	H6: Refactoring was preferentially applied to the modules with high test adequacy.	Confirmed
Layer	H7: Preferential refactoring is correlated with reduction in the layer number.	Rejected

preferential refactoring— applying refactorings more frequently to a module, relative to the frequency of regular changes

Refactoring Models

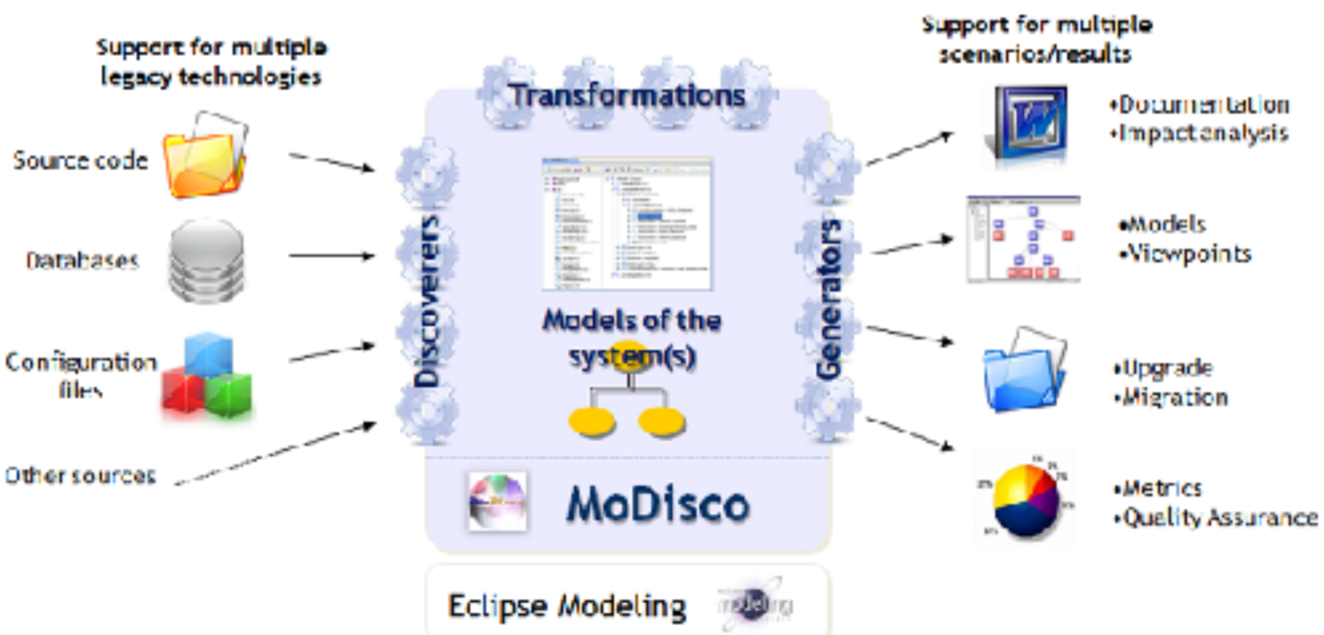
- Refactoring = Model Transformation
- Refactoring models — we're working e.g. on UML diagrams.
- Refactoring *through* models — we don't work on the source, but an intermediate representation. Example: class diagram derived from Java code.
- Refactoring *meta*models and instances.

Refactoring in Model-Driven Software Engineering

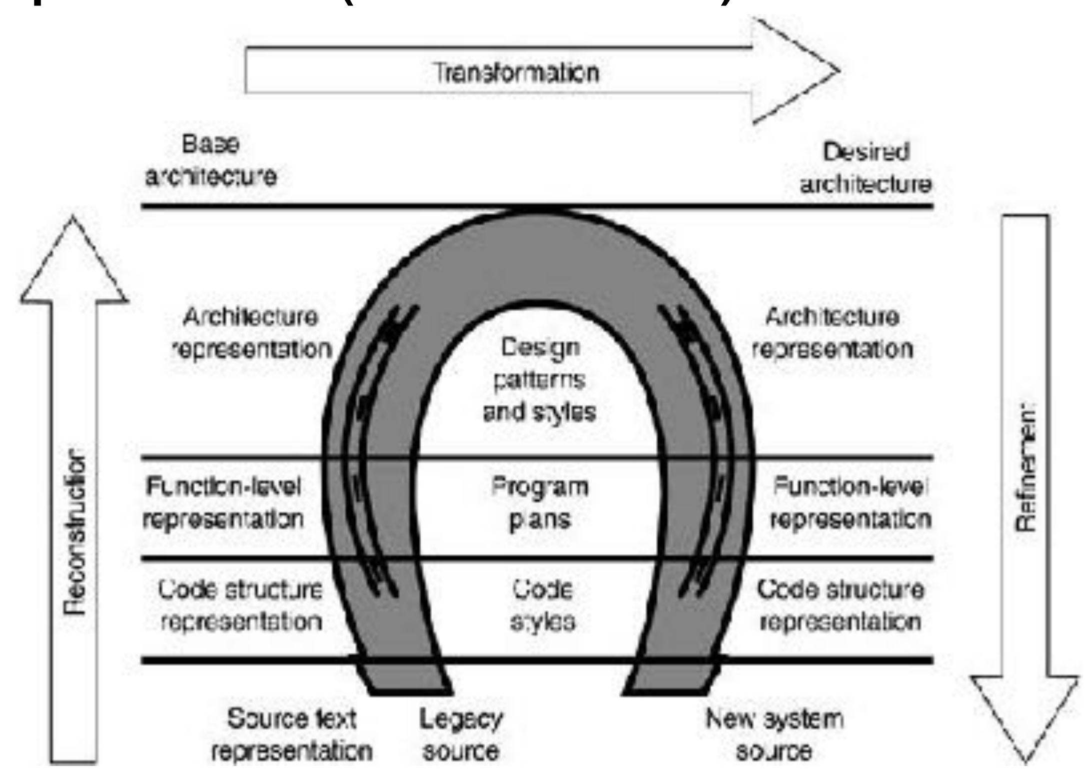
- Based on Graph Theory
- Higher level of abstraction, hence simpler reasoning
- Refactorings as model-to-model transformations
- Applicable to ***general-purpose*** modelling languages (e.g. UML) and ***domain-specific*** modelling languages
- Applicable to low-level or big-scale refactoring (reverse engineering and modernisation)
- Easier specification and visualisation of results

Reverse Engineering of Legacy Systems

- MoDisco Text-to-Model transformations



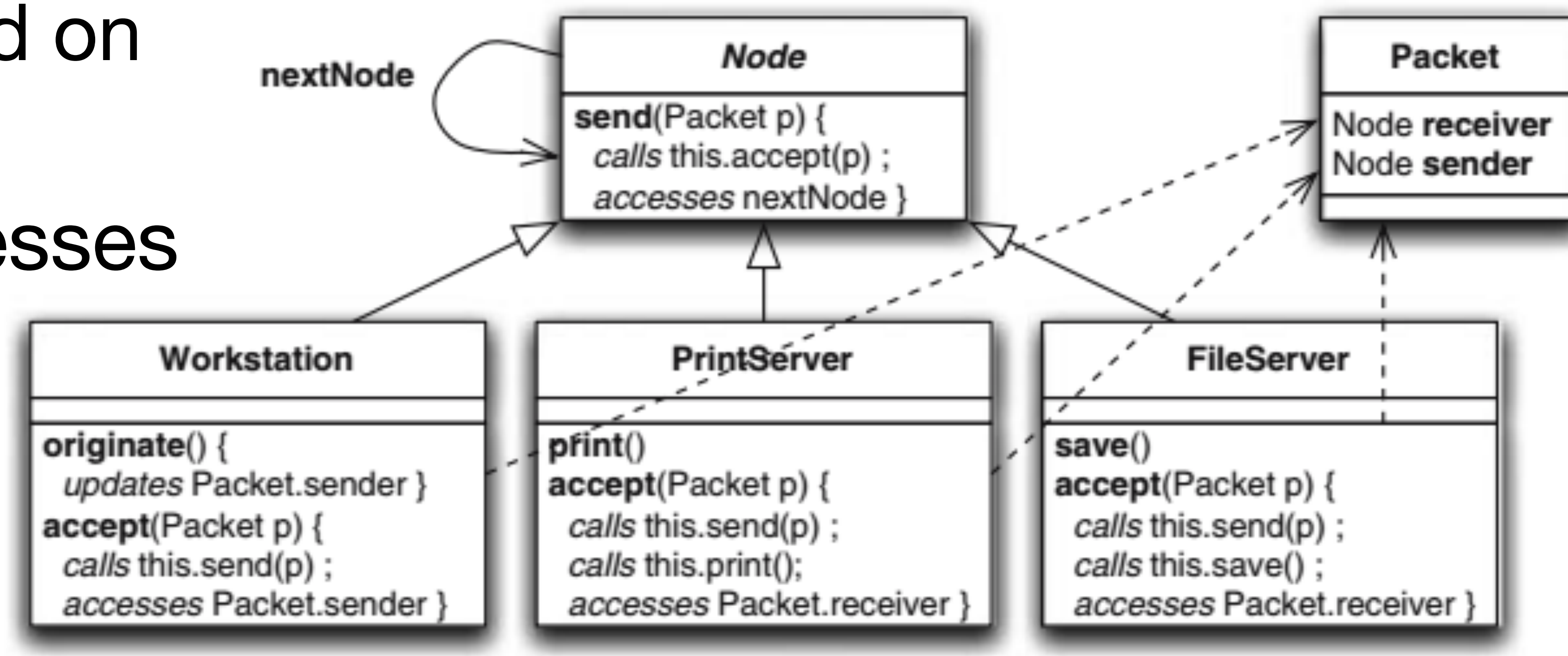
- Three-phase reengineering process (horseshoe)



Source: Erik Philippus

Model-based Refactoring of Source Code

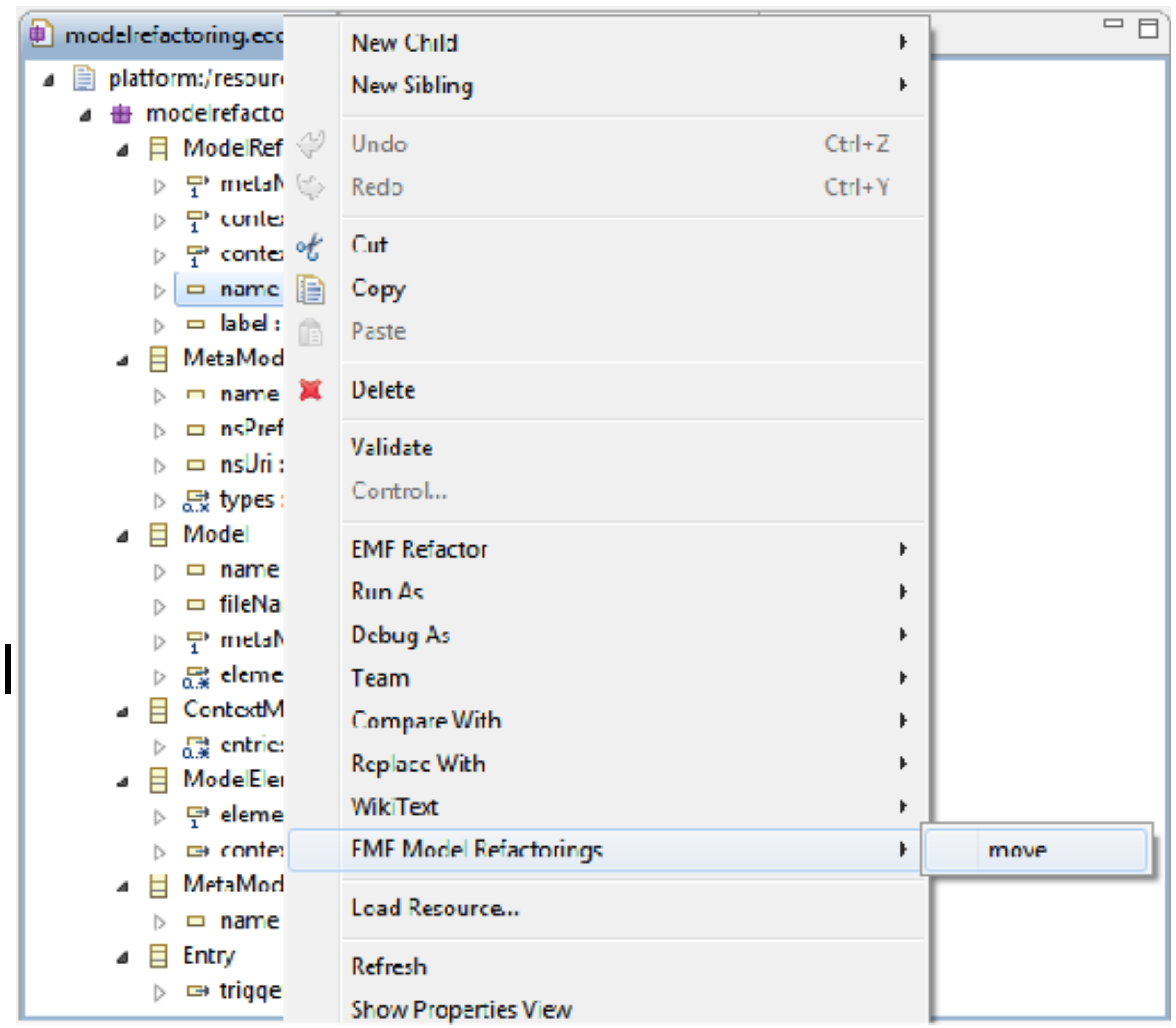
- Based on the analysis of annotated UML models
- Coordination of refactoring operations based on dependencies
  - Variable accesses
  - Method calls



T. Mens, G. Taentzer, O. Runge: "Analysing refactoring dependencies using graph transformation." *Software and Systems Modeling,* 2007

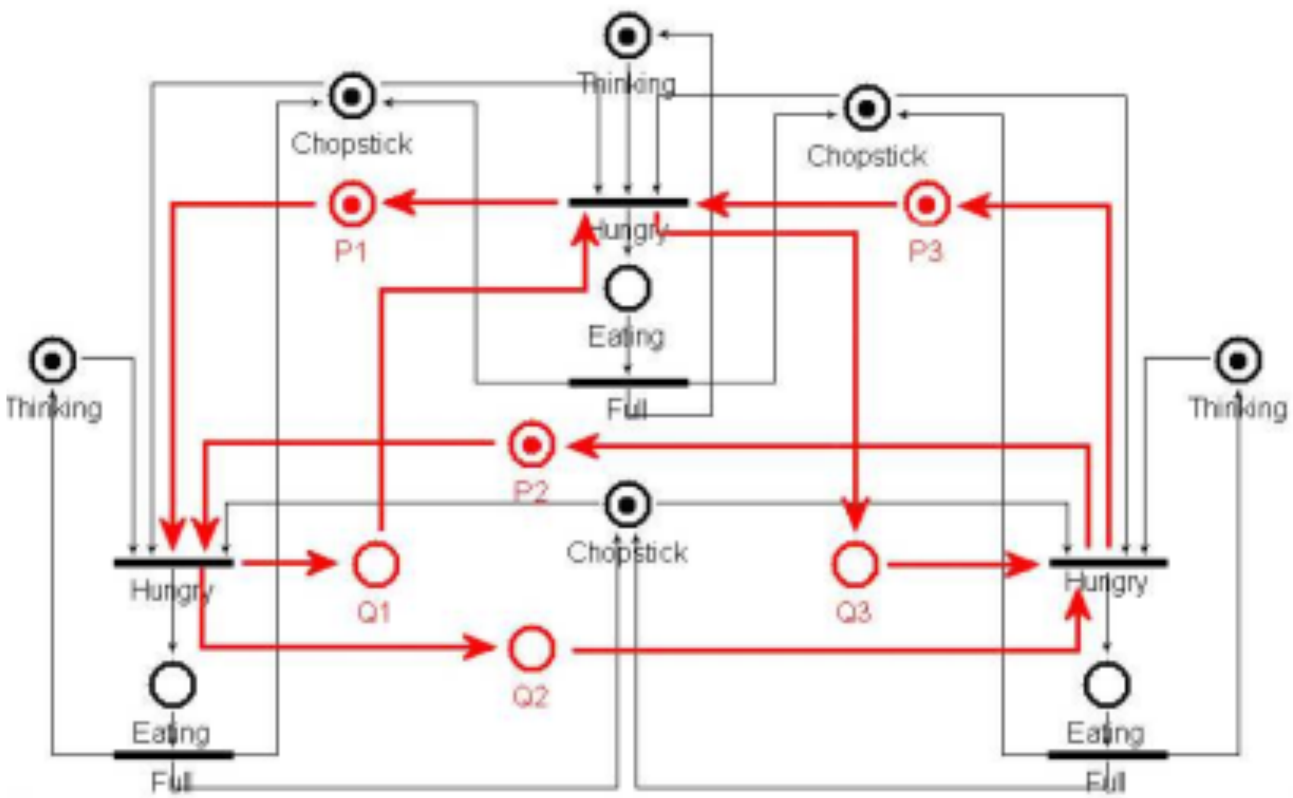
Refactoring of General-Purpose Languages

- Support in EMF through EMF Refactor
- Refactorings based on class diagrams (Ecore metamodels)
- Extensible framework for any EMF-based model



Source: eclipse.org

Refactoring of DSMLs

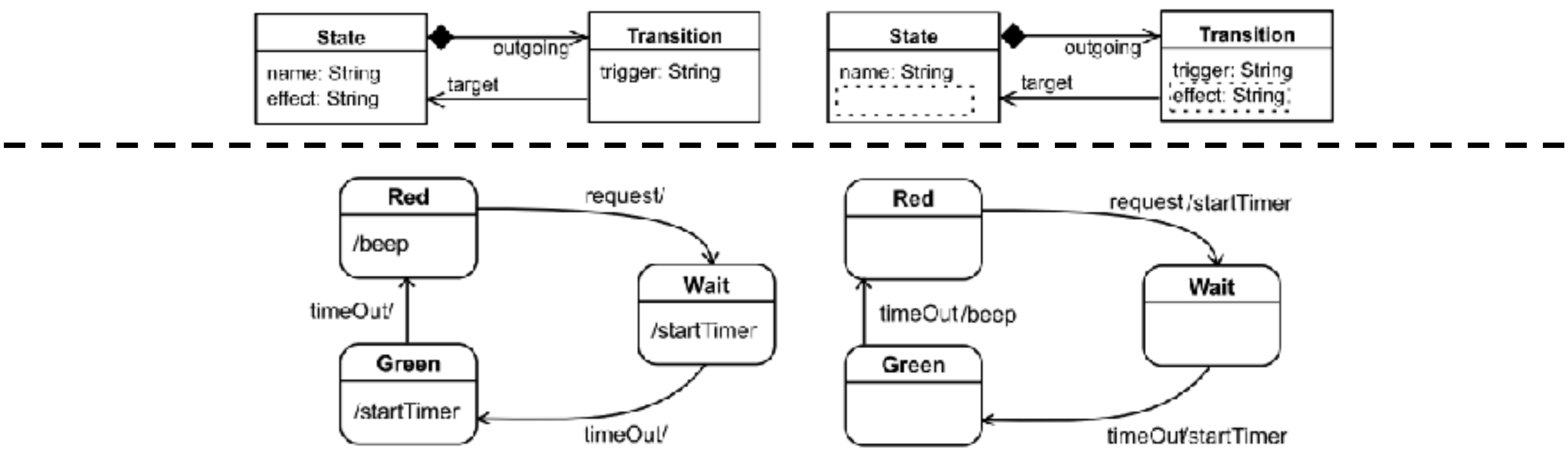



```
strategy freeStarvation()
{
    declare dstList:modelList;
    declare dst1,dst2,p,q:model;
    declare static num:Integer;
    dstList:=findOutConnections();
    assert(dstList.size()==2);
    dst1:=dstList.get(0);  dst2:= dstList.get(1);
    p:=createModel("InitMarker","P"+intToStr(num));
    q:=createModel("Place", "Q" + intToStr(num));
    num:=num + 1;
    addConnection(dst1,p);  addConnection(p,dst2);
    addConnection(dst2,q);  addConnection(q,dst1);
}
```

J. Zhang, Y. Lin, J. Gray: "Generic and domain-specific model refactoring using a model transformation engine", 2005

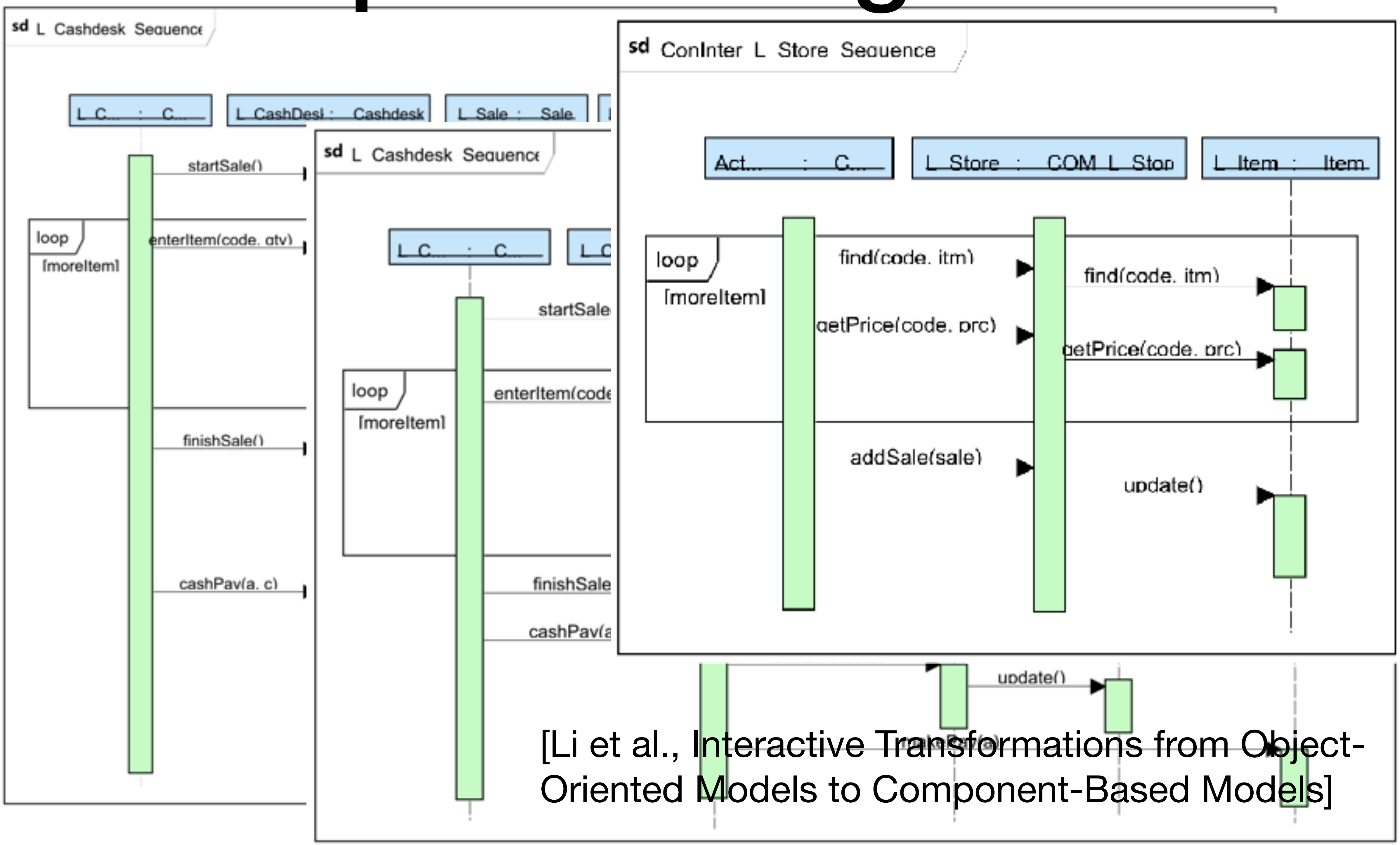
Refactoring of Metamodels and Their Instances

- AKA co-evolution AKA model migration



Mantz et al. “Co-evolving meta-models and their instance models: A formal approach based on graph transformation”, 2015

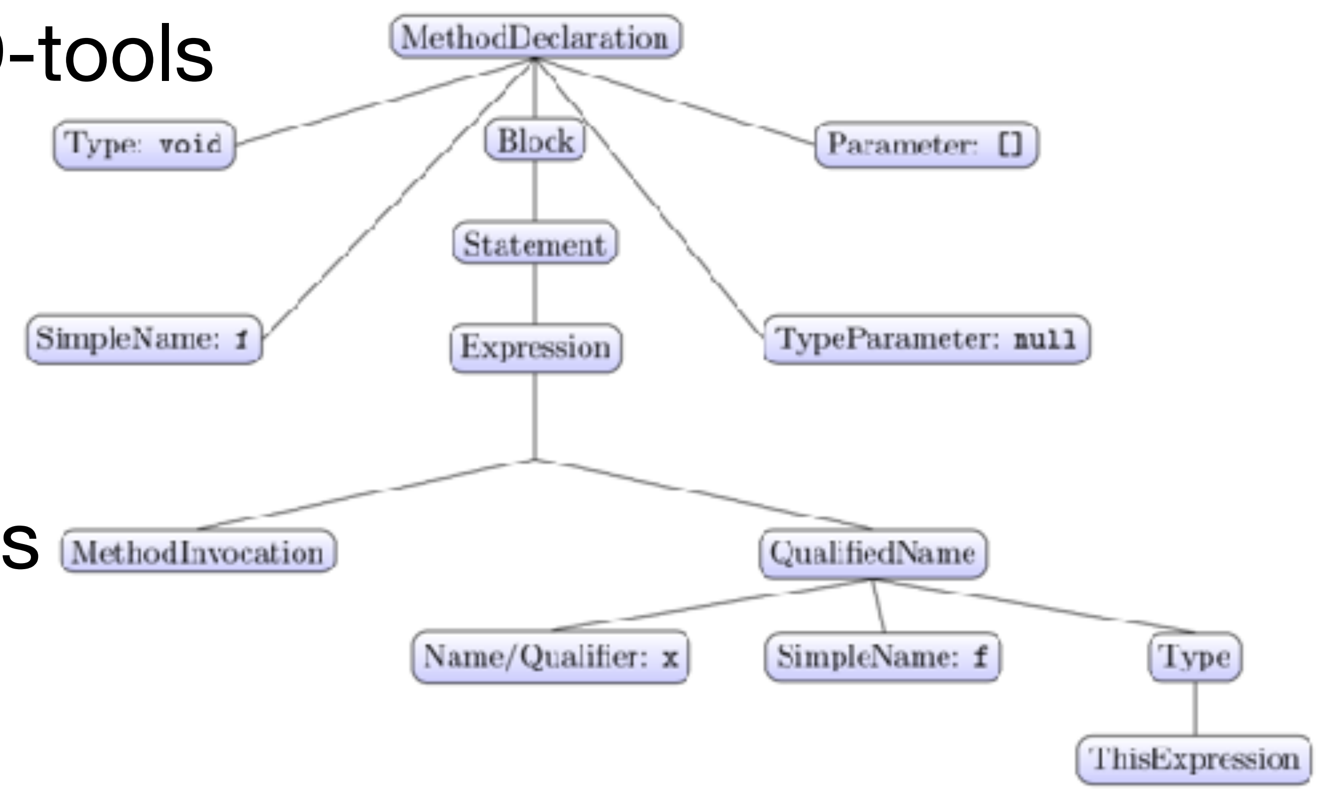
Refactoring an UML Sequence Diagram



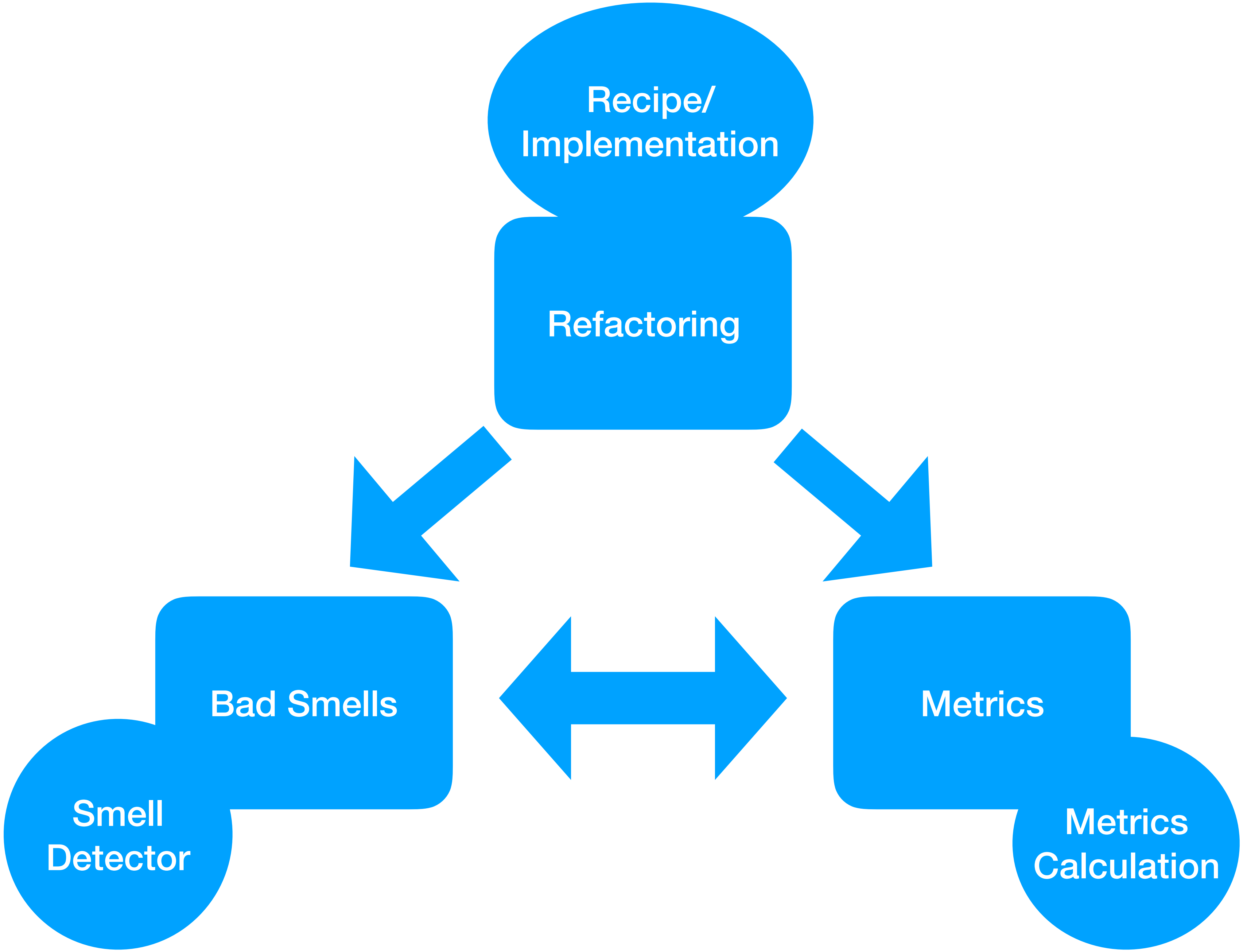
[Li et al., Interactive Transformations from Object-Oriented Models to Component-Based Models]

Refactorings = Model Transformation?

- Many existing approaches for graph (tree!) -based models
- AST = model
- Can use OCL and other MDD-tools to describe transformations
- Limitation:
  - many syntactical elements → complex transformations
  - formal reasoning difficult
    - *refactoring always correct* vs. *instance of refactoring correct* (with proof obligations)

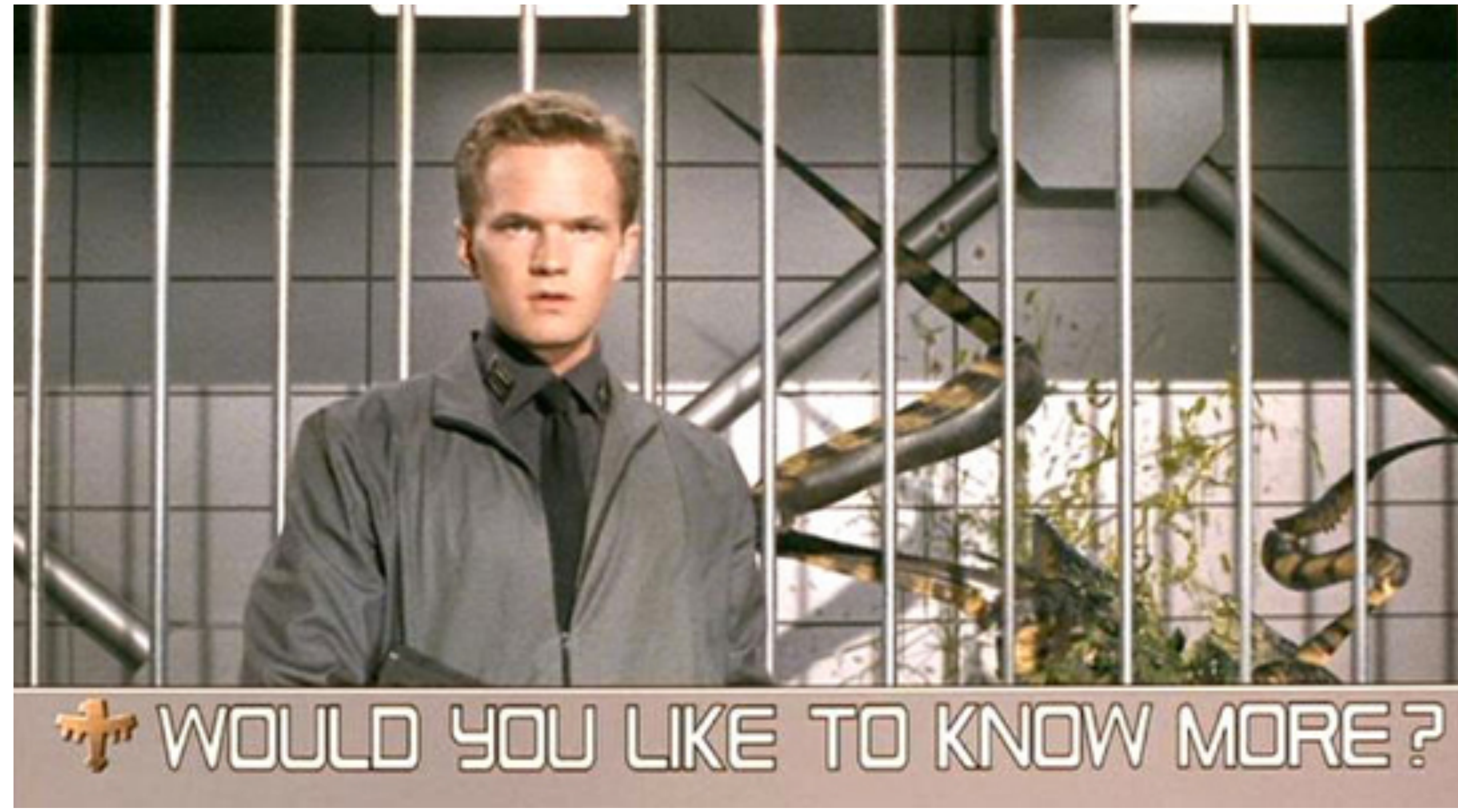


Summary


Recipe/
Implementation
Refactoring
Bad Smells
Metrics
Smell
Detector
Metrics
Calculation

Interested?

- Refactorings are an important software engineering topic.
- Tool support for refactoring *always* needs improvement.
- Related topics:



  - *compiler construction* (to work with programs as input: syntax & semantics, grammars & parsing, ASTs, typing rules)
  - *logic/discrete maths* (∀,∃,∊,… to read & write specifications)
  - optional: *static analysis* (information flow analyses etc., also useful for security analysis)
  - most of all: interest in *coding*!

Bibliography (1)

Foundations:

- W. F. Opdyke. Refactoring object-oriented frameworks. Technical Report GAX93-05645, University of Illinois at Urbana-Champaign, 1992.
- D. Roberts, J. Brant, and R. Johnson. A refactoring tool for SmallTalk. Theory and Practice of Object Systems, 3(4):253–263, 1997.
- M. Fowler. Refactoring: improving the design of existing code. Addison-Wesley, 1999.
- E. Gamma, J. Helm, R. Johnson, R. Vlissides: “Design Patterns: Elements of Reusable Object-Oriented Software”, 1994
- J. Kerievsky: “Refactoring to patterns”, Addison-Wesley, 2005
- M.O’Keeffe and M.Ó.Cinnéide. Search-based refactoring: An empirical study. J. of Software Maintenance and Evolution, 20(5):345–364, 2008.
- P. Pirkelbauer, D. Dechev, B. Stroustrup: Source Code Rejuvenation Is Not Refactoring. SOFSEM 2010
- M. Kim, T. Zimmermann, N. Nagappan: A field study of refactoring challenges and benefits. Intl. Symp. on the Foundations of Softw. Eng. ACM, 2012
- G.C. Murphy, M. Kersten, L. Findlater: How are Java software developers using the Elipse IDE? IEEE Software 23(4), 2006.
- E. Tempero, T. Gorschek, L. Angelis: Barriers to Refactoring. Communications of the ACM, Vol. 60 No. 10, 2017

Bibliography (2)

“Modern” Reading:

- M. Vakilian, N. Chen, S. Negara, B. A. Rajkumar, B. P. Bailey, and R. E. Johnson. Use, disuse, and misuse of automated refactorings. In 34th Intl. Conf. on Software Engineering (ICSE 2012). IEEE, 2012.
- E. Murphy-Hill, C. Parnin, and A. P. Black. How we refactor, and how we know it. Software Engineering, IEEE Transactions on, 38(1):5–18, 2012.
- J. Brant and F. Steimann. Refactoring tools are trustworthy enough and trust must be earned. IEEE Software, 32:80–83, 2015.
- G. Soares, B. Catao, C. Varjao, S. Aguiar, R. Gheyi, T. Massoni: Analyzing Refactorings on Software Repositories. SBES 2011
- G. Soares, R. Gheyi, D. Serey, and T. Massoni. Making program refactoring safer. IEEE Software, 27(4):52–57, 2010.
- M. Mongiovi, R. Gheyi, G. Soares, L. Teixeira, and P. Borba. Making refactoring safer through impact analysis. SCP, 93:39–64, 2014.
- T. Mens, G. Taentzer, and O. Runge. Analysing refactoring dependencies using graph transformation. Software & Systems Modeling, 6(3):269–285, 2007.
- M. Schäfer and O. de Moor. Specifying and implementing refactorings. In Object Oriented Programming: Systems, Languages, and Applications (OOPSLA) ’10. ACM, 2010.
- A. M. Eilertsen, A. H. Bagge, and V. Stolz. Safer refactorings. In Proc. of the Intl. Symp. On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA), LNCS. Springer, Oct 2016.

Bibliography (3)

- F. Medeiros, M. Ribeiro, R. Gheyi, S. Apel, C. Kästner, B. Ferreira, L. Carvalho, and B. Fonseca: Discipline Matters: Refactoring of Preprocessor Directives in the #ifdef Hell. Transactions on Software Engineering (2017).
- A. Garrido and R. Johnson. 2002. Challenges of Refactoring C Programs. In Proceedings of the 5th International Workshop on Principles of Software Evolution. ACM, 6–14.
- A. Garrido and R. Johnson. 2013. Embracing the C Preprocessor during Refactoring. Journal of Software: Evolution and Process 25, 12 (2013), 1285–1304.
- H. Wright, D. Jasper, M. Klimek, C. Carruth, and Z. Wan. Large-scale automated refactoring using ClangMR. Intl. Conf. on Software Maintenance. IEEE, 2013.
- E.L.G. Alves, T. Massoni, P.D. de Lima Machado: Test coverage of impacted code elements for detecting refactoring faults: An exploratory study. J. Systems and Software 123, 2017.
- D. Li, X. Li, Z. Liu, V. Stolz: Interactive Transformations from Object-Oriented Models to Component-Based Models. FACS 2011, LNCS, Springer 2011.
- F. Mantz, et al.: Co-evolving meta-models and their instance models: A formal approach based on graph transformation. Science of Computer Programming 104 (2015): 2-43.
- J. Zhang, Y. Lin, J. Gray: Generic and domain-specific model refactoring using a model transformation engine. Model-driven Software Development 23 (2005).