CSSE 490 Model-Based Software Engineering: Transformation Systems

Shawn Bohner

Office: Moench Room F212

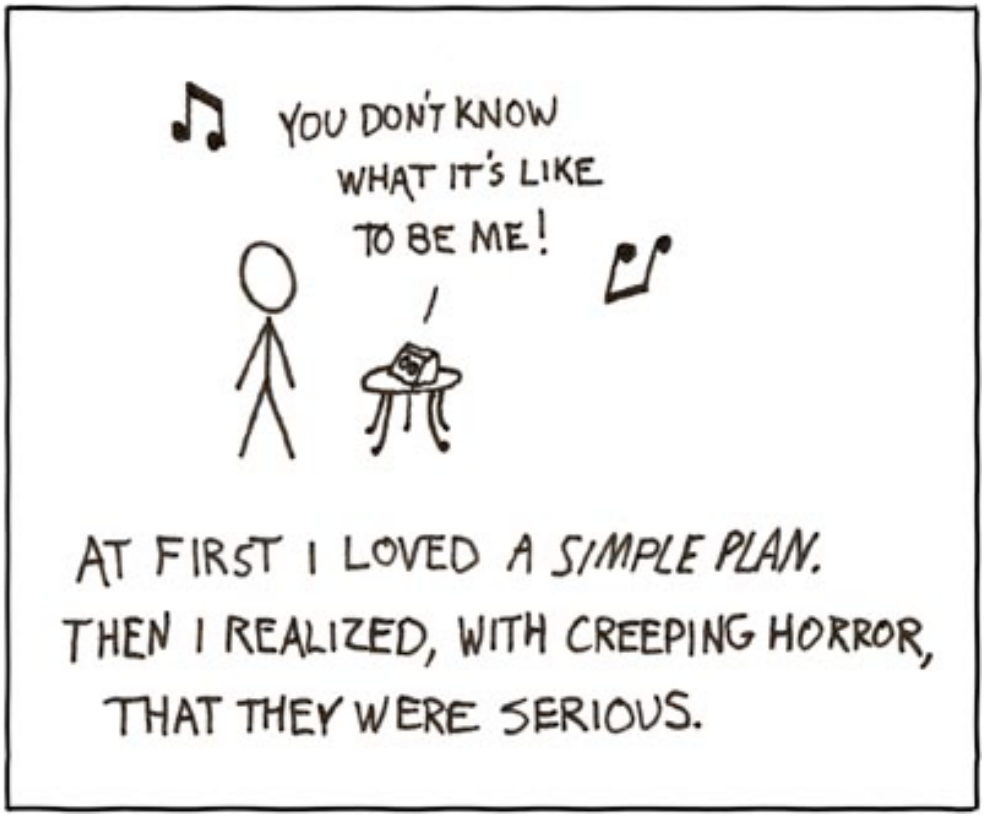
Phone: (812) 877-8685

Email: bohner@rose-hulman.edu

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Plan for Today

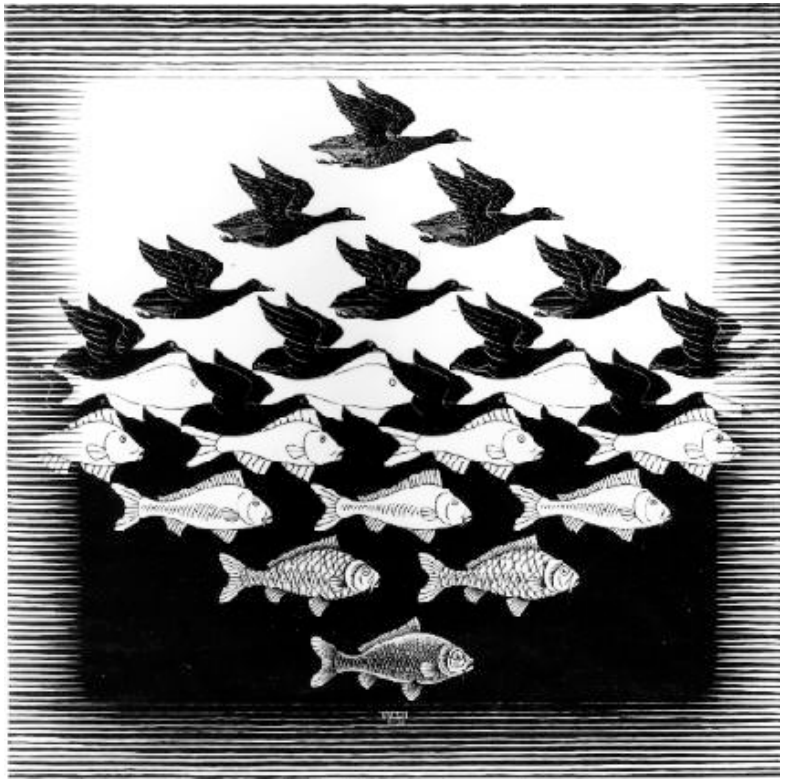
- FacePamphlet Demo and Discussion
- Continue QVT and introduce Eclipse Modeling Project
- Examine Transformation Systems (if time)
- Homework Assignments



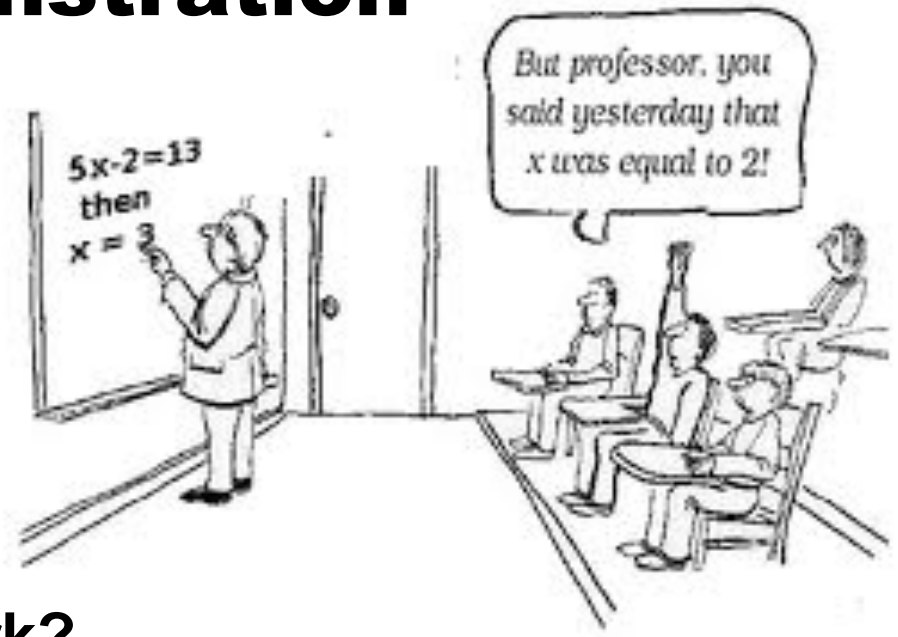
Learning Outcomes: Transformations

Define transformation rules for abstraction and refinement.

- Describe QVT transformations
- Explore Model to Model transform
- Introduce Eclipse Modeling Project



Milestone 1 - Demonstration



- FacePamphlet – a subset of Facebook
- Some questions to answer in the demo:
  - Do the main features work?
  - How is the code organized for separating concerns? Major capabilities? Platforms → GUI? Database?
  - Are the artifacts able to be organized into a repository?

What are some examples of rewrite rules you might want to employ in generating software from models?

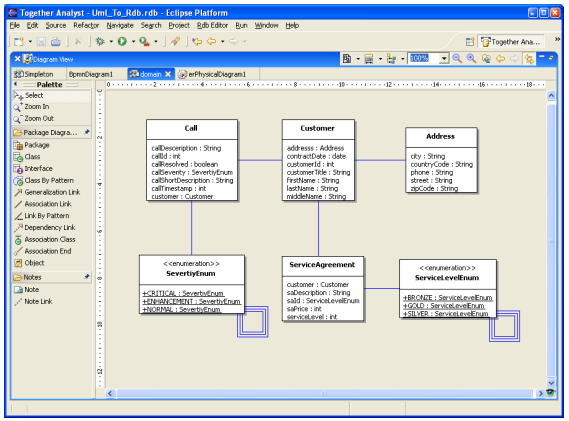
- Think for a 15.332 seconds…
- Turn to a neighbor and discuss it for a minute

Recall: Query-View-Transformation

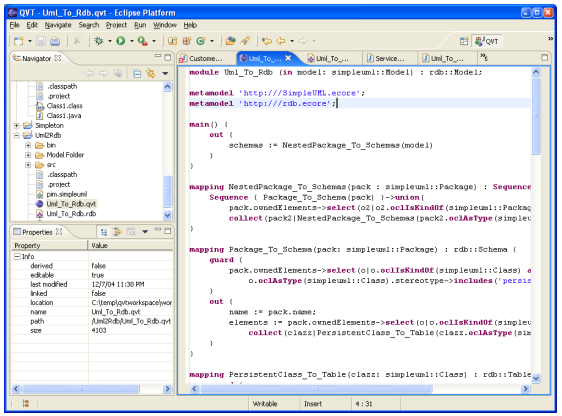
- QVT specification is the heart of Model Driven approaches
- **Queries** take a model as input and select specific elements from that model
- **Views** are models that are derived from other models
- **Transformations** take a model as input and update it or create a new model

Recall: UML to RDB Example

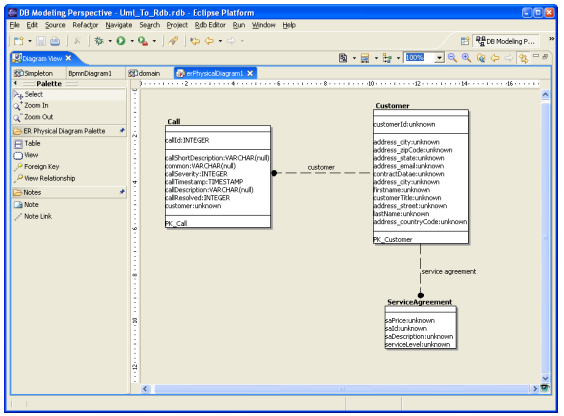
UML Class model → Relational Data Model



Model



Query



View

QVT can be used to Transform…

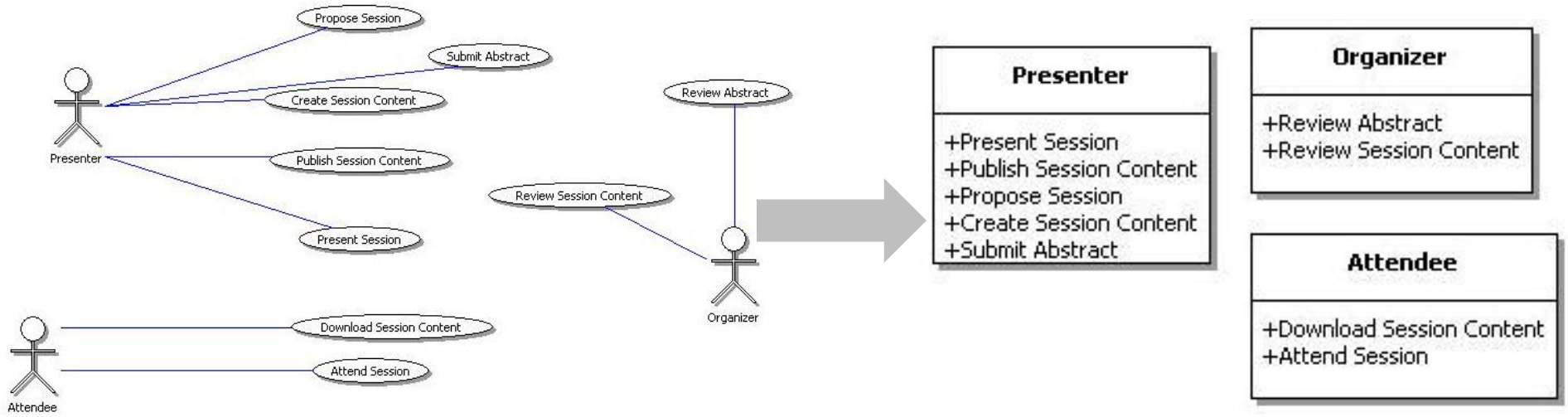
- Business Process Model → Object Model (PIM to PIM)
- Analysis Object Model → Business Object Model (PIM to PIM)
- Object Model → Data Model (PIM to PIM or PIM to PSM)
- Object Model → Detailed Object Model (PIM to PSM)

QVT: Transformations

- QVT provides language to implement model-to-model transformations
- QVT supports OCL 2.0 expressions
  - We will examine OCL later
- Query libraries
  - Reusable libraries of QVT mappings
- Traceability
  - Automated traceability when executing mapping
- Extensible
  - QVT can call custom Java methods

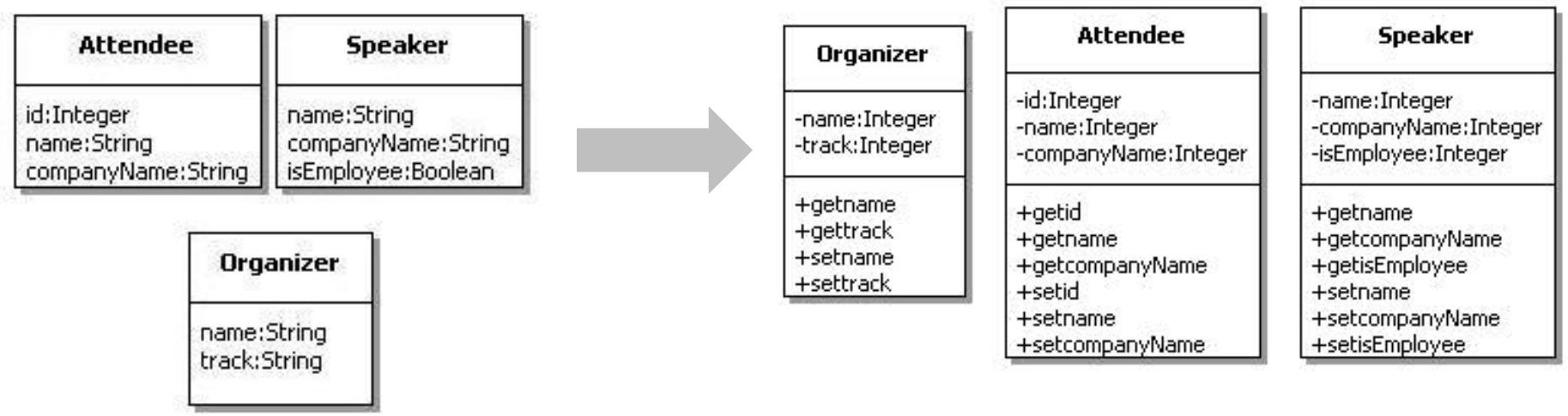
Transforming a Use Cases to Classes

- This example automates the construction of a set of use case realization classes
- A simple 1-to-1 mapping from Actor to Class
  - Use Cases owned by the Actor are created as Operations within the Class



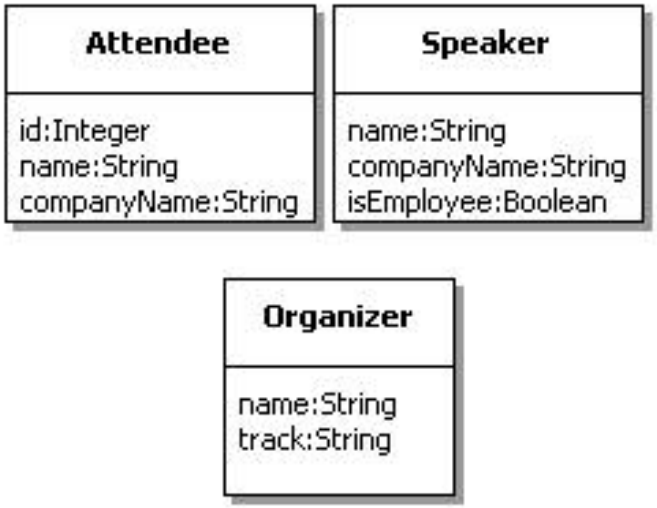
Transforming a PIM to a PSM

- As a basic step toward elaborating the PSM, this QVT simply adds scaffolding code (e.g., getters and setters)



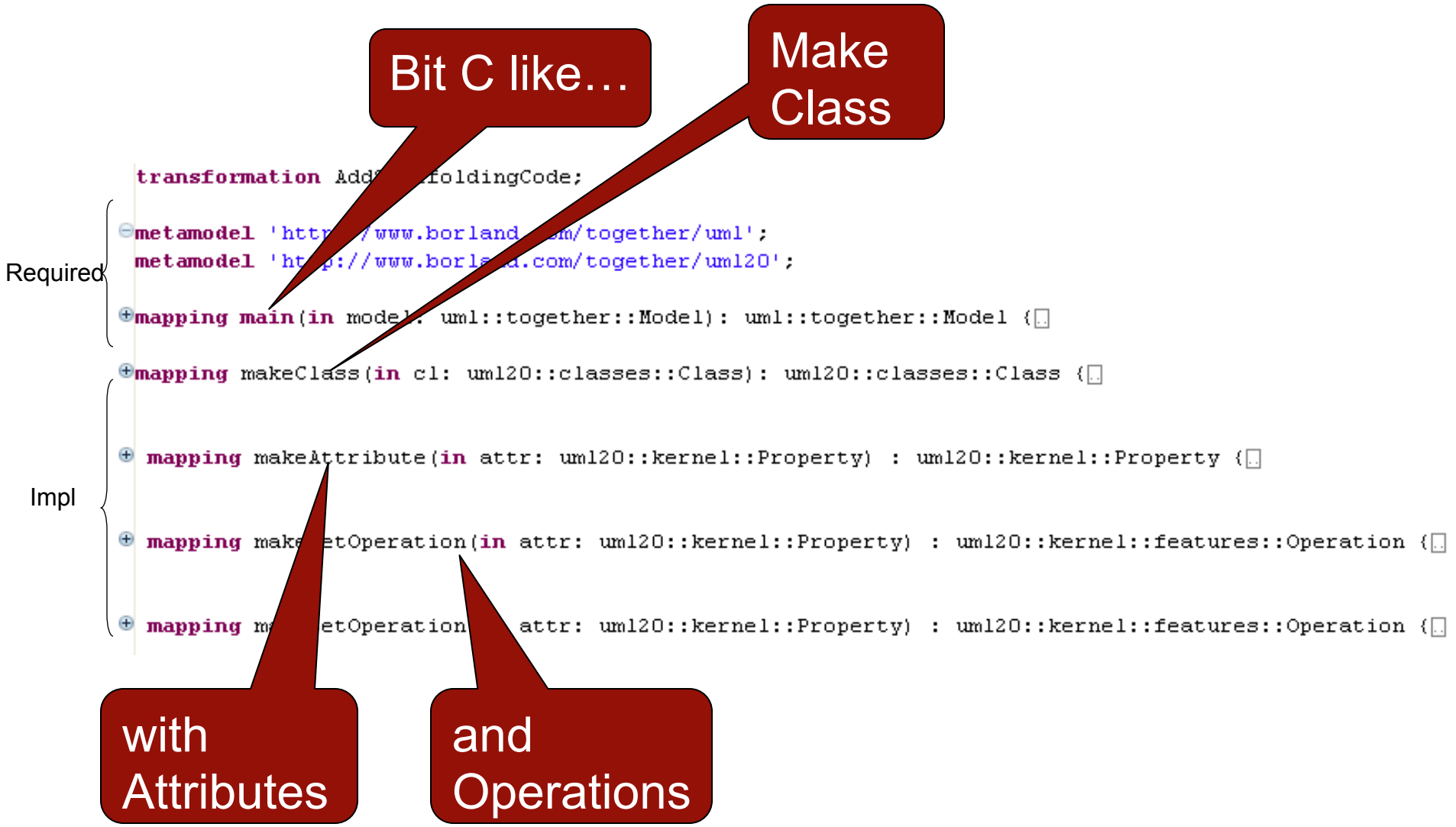
Simple QVT Example: PIM to PSM

- PIM: Three classes and a few attributes…



- QVT: AddScaffoldingCode
  - Simply add getter and setter methods

AddScaffoldingCode.qvt



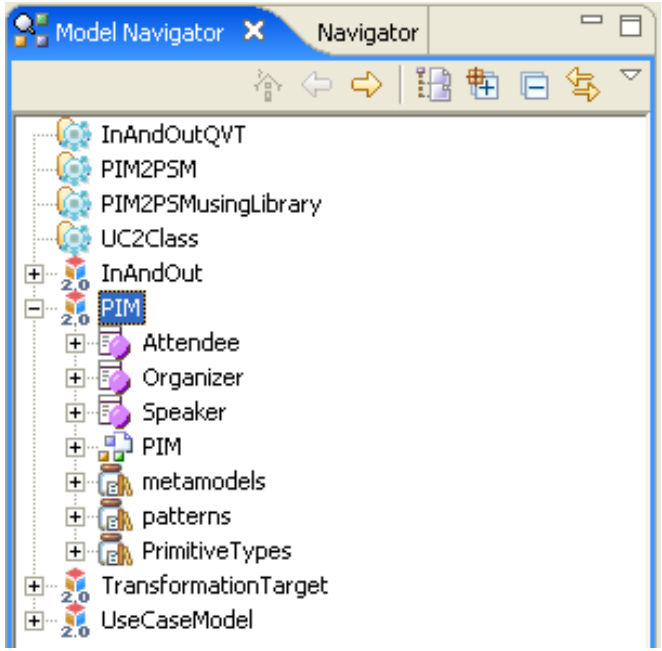
Control Flow in AddScaffoldingCode.qvt

```
transformation AddScaffoldingCode;

metamodel 'http://www.borland.com/together/uml';
metamodel 'http://www.borland.com/together/uml20';

mapping main(in model: uml::together::Model): uml::together::Model {
    object {
      ownedMembers := model.ownedMembers->select(it|it.oclIsTypeOf(uml20::classes::C
    }
}
mapping makeClass(in cl: uml20::classes::Class): uml20::classes::Class {
  init {
   var attrs := cl.attributes;
  }
  object {
   name := cl.name;
   description := 'put your description here...';
   attributes := attrs->collect(a | makeAttribute(a))->asOrderedSet();
   ownedOperations := attrs->collect(a | makeGetOperation(a))->asOrderedSet();
   ownedOperations += attrs->collect(a | makeSetOperation(a))->asOrderedSet();
  }
 }
   description := 'this is a generated getter';
  }
 }

 mapping makeSetOperation(in attr: uml20::kernel::Property) : uml20::kernel::features::Operation {
 object{
  name := 'set' + attr.name;
  visibility := uml::kernel::VisibilityKind::PUBLIC;
  description := 'this is a generated setter';
  }
```

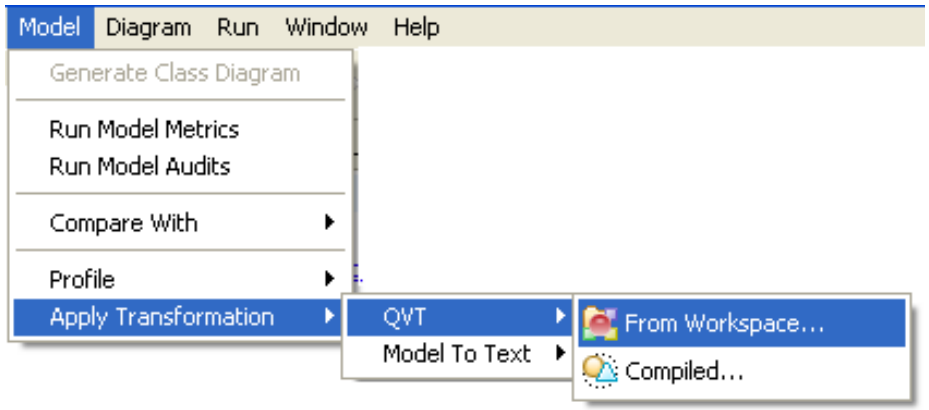
Mapping methods do most of the work

Applying QVT transformations 1/4

- Select input model

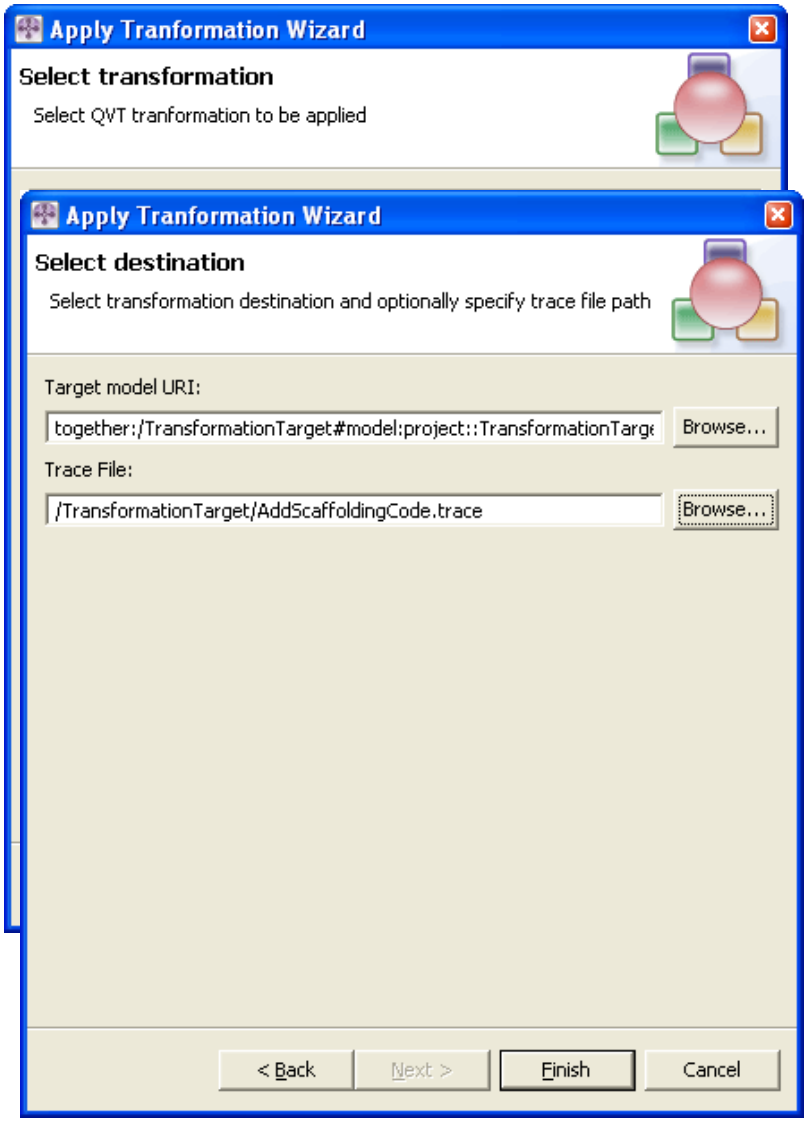


- Choose menu option



Applying QVT Transformations 2/4

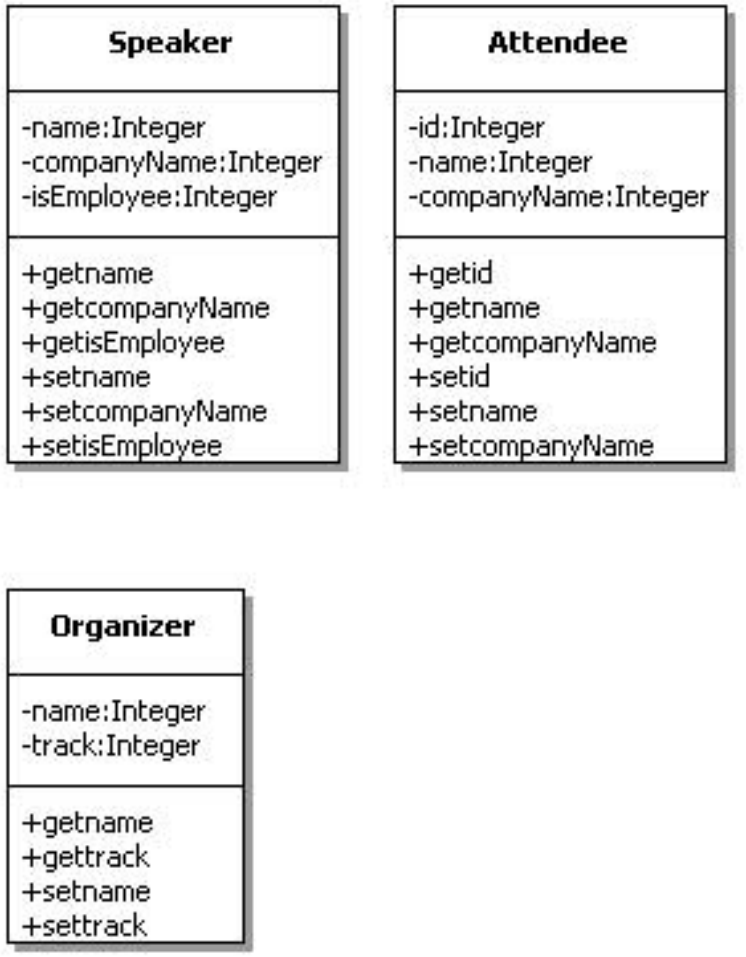
- Select QVT to use



- Select target model
  - Optionally create trace file

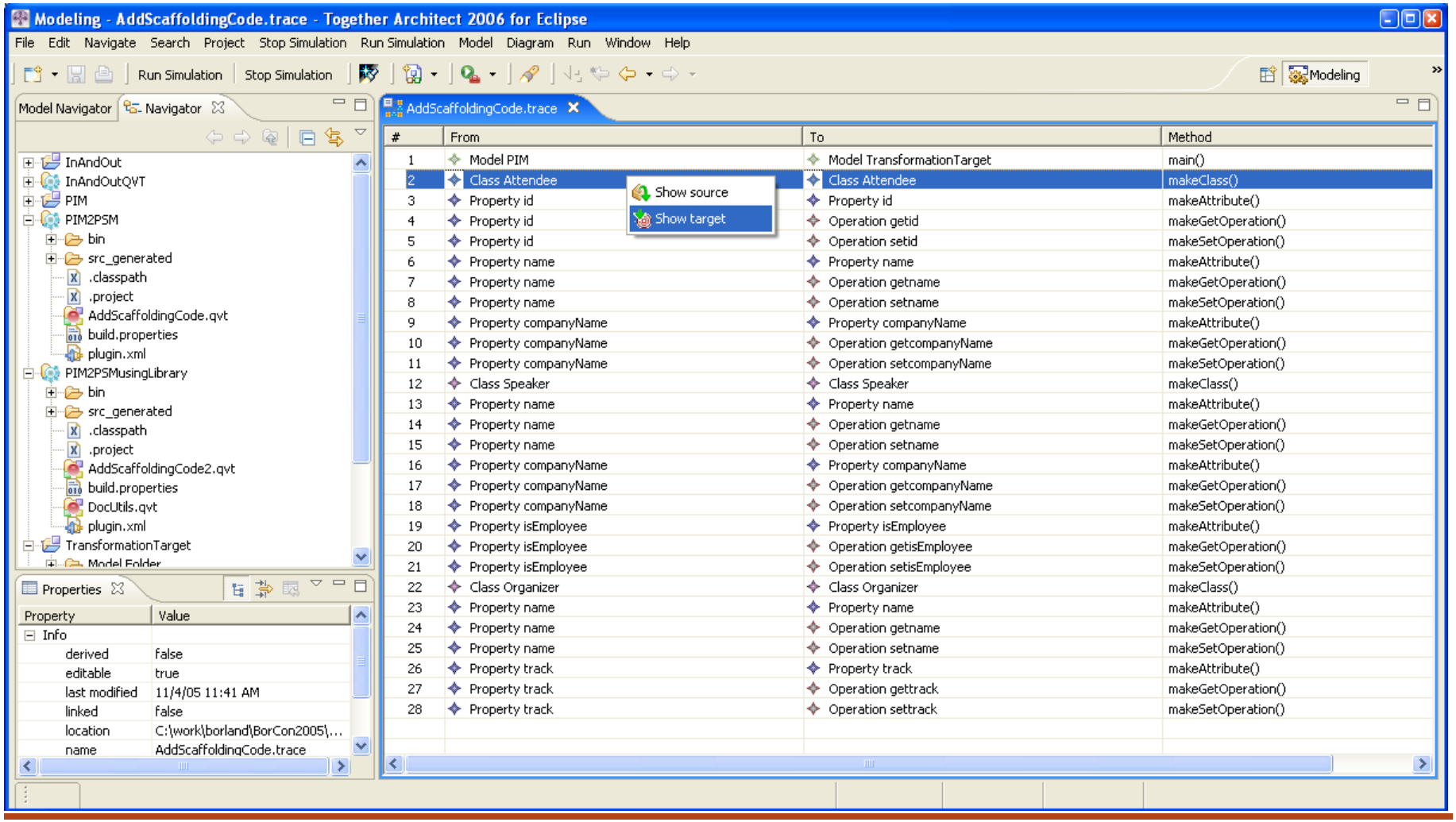
Applying QVT Transformations 3/4

- Transformation result
  - note added getter/setter methods

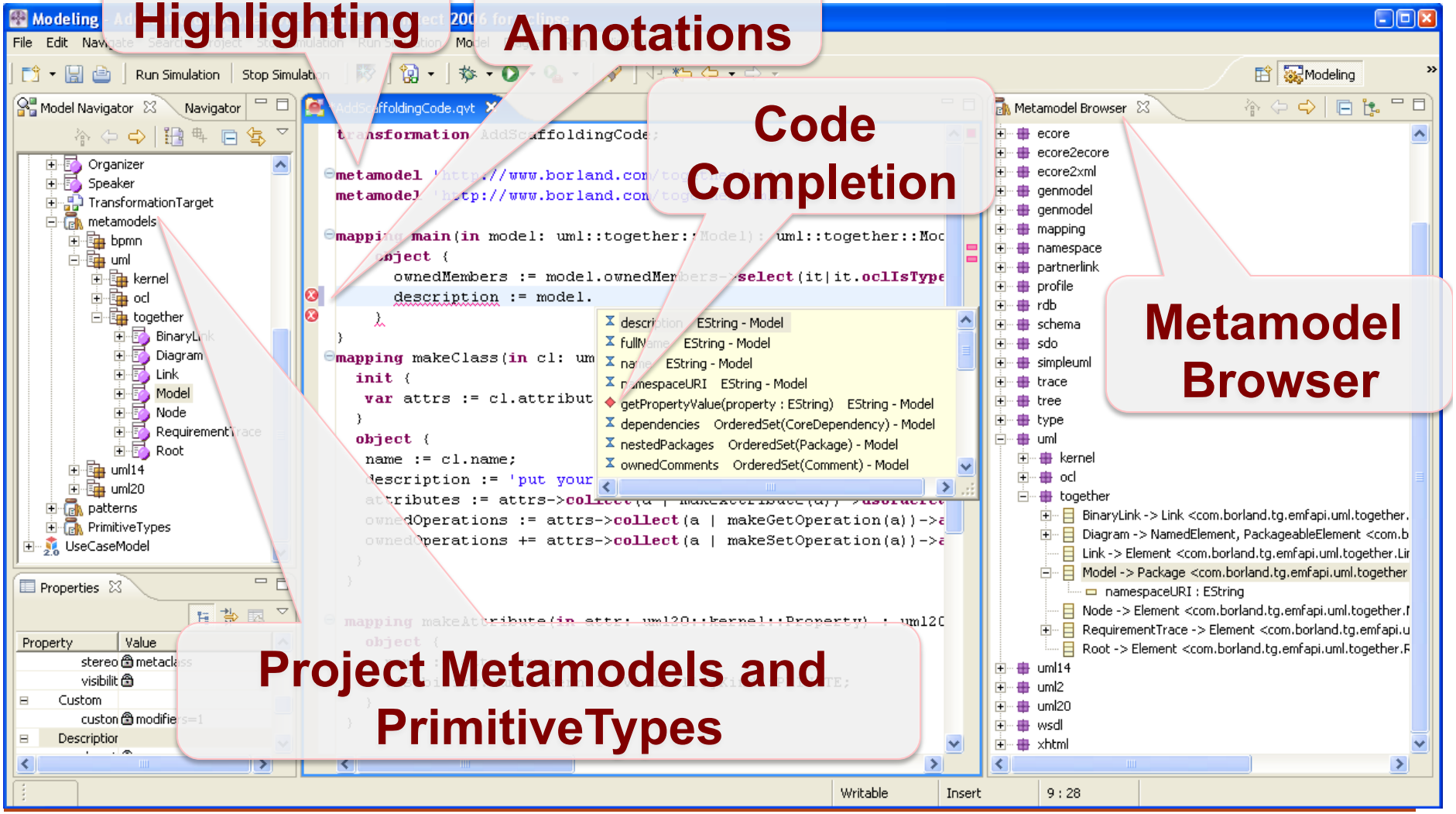


Applying QVT Transformations 4/4

Trace file view



Example MBSE Tools: Editor, Model Navigator, and Metamodel Browser



ROSE-HULMAN INSTITUTE OF TECHNOLOGY

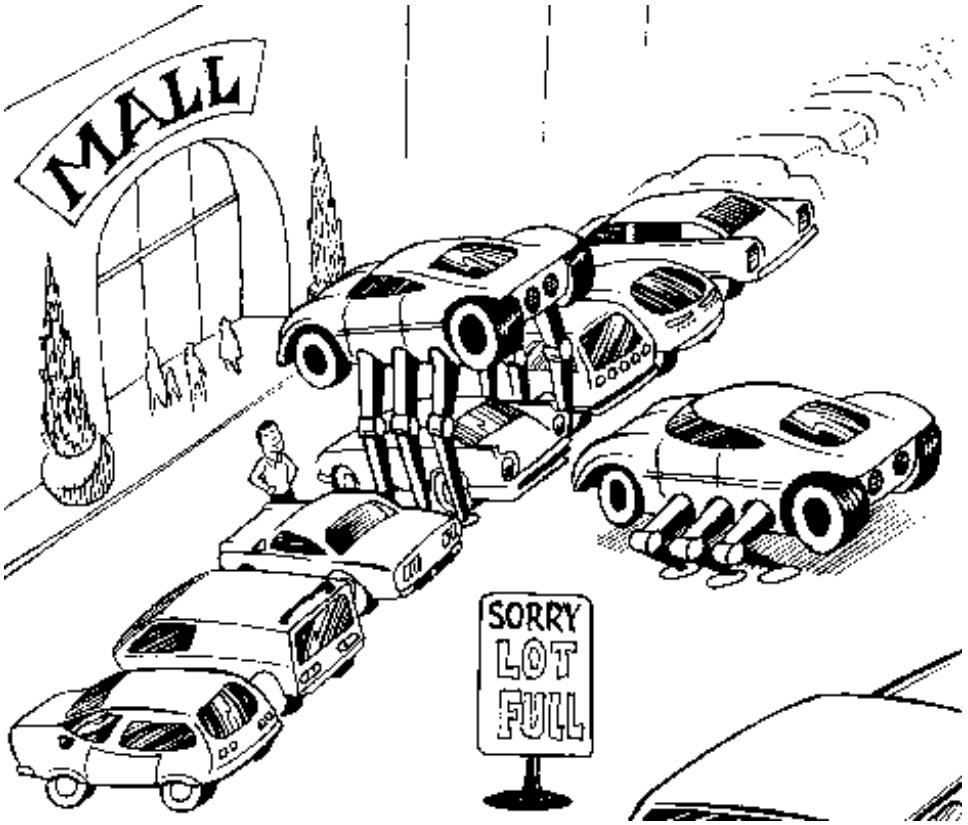
Eclipse Modeling Project

- Promotes model-based engineering technologies within the Eclipse community

- Provides a unified set of modeling frameworks, tooling, and standards implementations
  - http://www.eclipse.org/modeling/
- QVT and other MBE resources
  - http://www.eclipse.org/m2m/

Benefits of Using Transforms

- Intermediate work products vanish due to clear value of all models
- Repeatable, high-quality approach to software design /development



- Automated traceability between models, queries, transformations and views

Homework and Milestone Reminders

- Read Feature-Based Transformation Approach Paper (via schedule page)
- Familiarize yourself with material on Eclipse Modeling Project

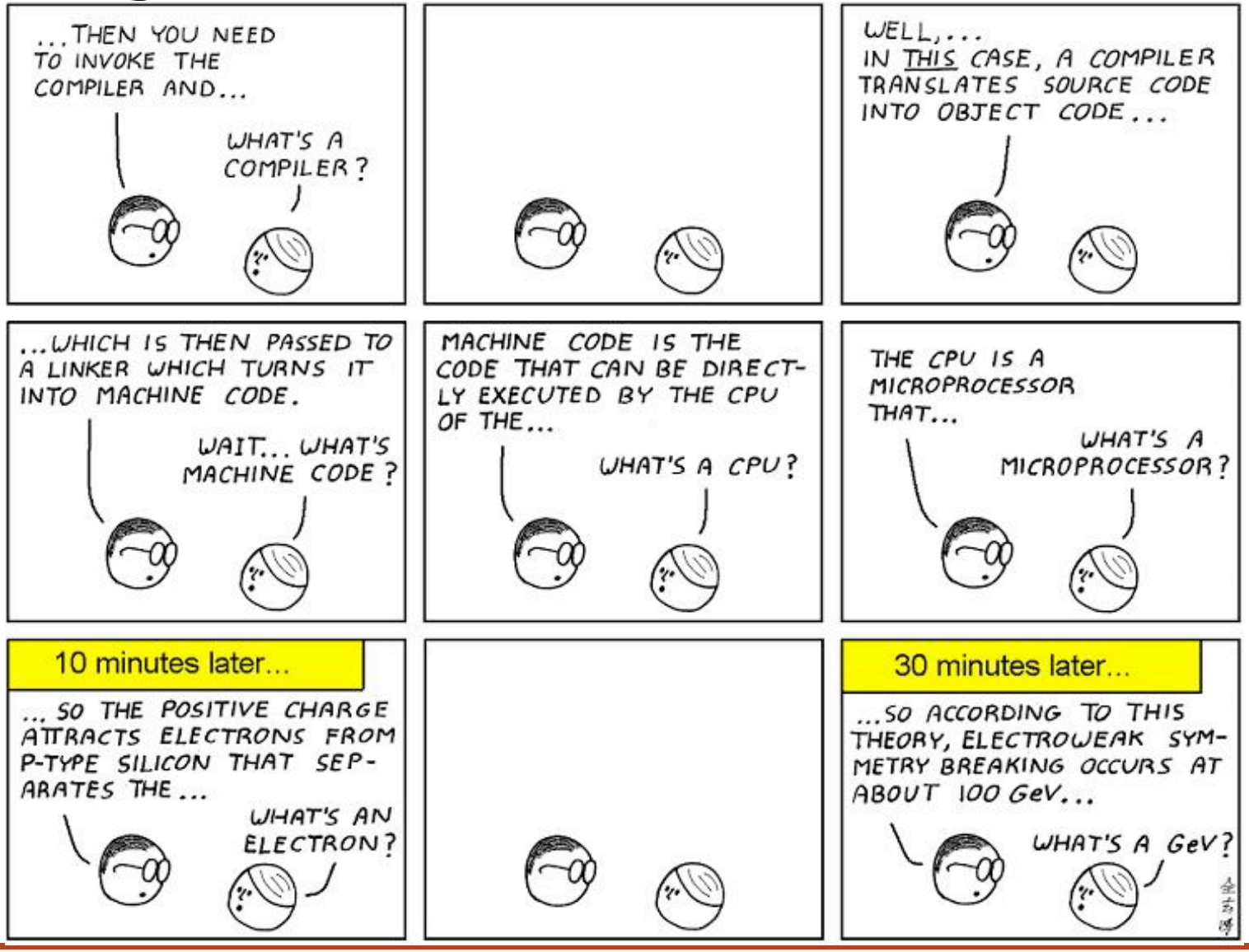
  http://www.eclipse.org/modeling/
- Let's talk tomorrow in more detail about Transformational Programming and Systems

Transformational Programming

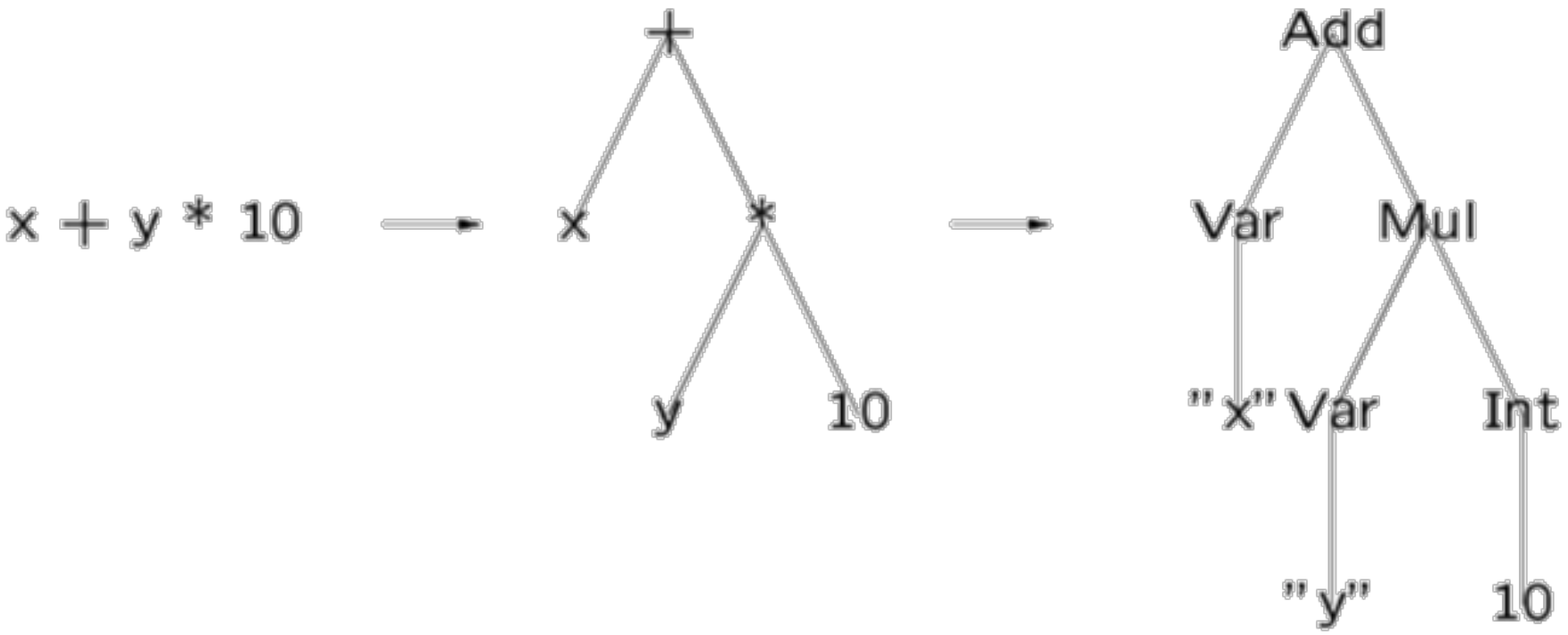
- Programming by successive application of transformation rules
- **Transformation** — a relation between two programs
- **Transformation rule** — mapping from one program to another that constitutes a correct transformation (e.g., equivalency)

- Guarantees that the final version of the program satisfies the initial formal specification

It may feel a little like this...



Program Representation



Describing Languages

- Terms can be used to describe arbitrary structured information
- A program corresponds to a subset of the set of all terms
- A signature describes a set of terms
  - Declaration of sort names

    `Sorts S1 … Sn`
  - Declaration of constructors

    `constructors`
    `C1 : S`
    `C2 : $S_1$ * … * $S_n$ → $S_0$`

Propositional Formulae 1/2

```
Module Group
Signatures
  Sorts Prop
  constructors
    False : Prop
    True : Prop
    Var : String → Prop          //Proposition Letter
    Not : Prop → Prop            //Negation
    And : Prop Prop → Prop       //conjunction
    Or : Prop Prop → Prop        //disjunction
    Impl : Prop Prop → Prop      //implication
    Eq: Prop Prop → Prop         //Equivalence
```

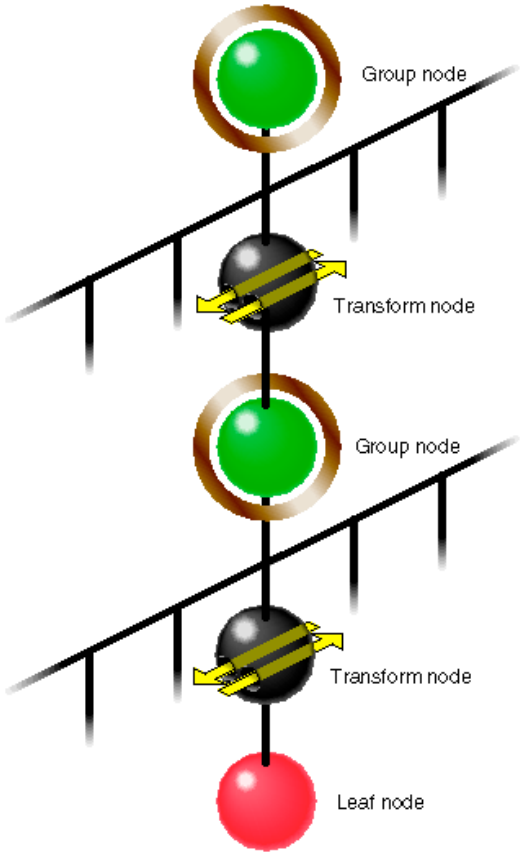
Propositional Formulae 2/2

- Example Terms

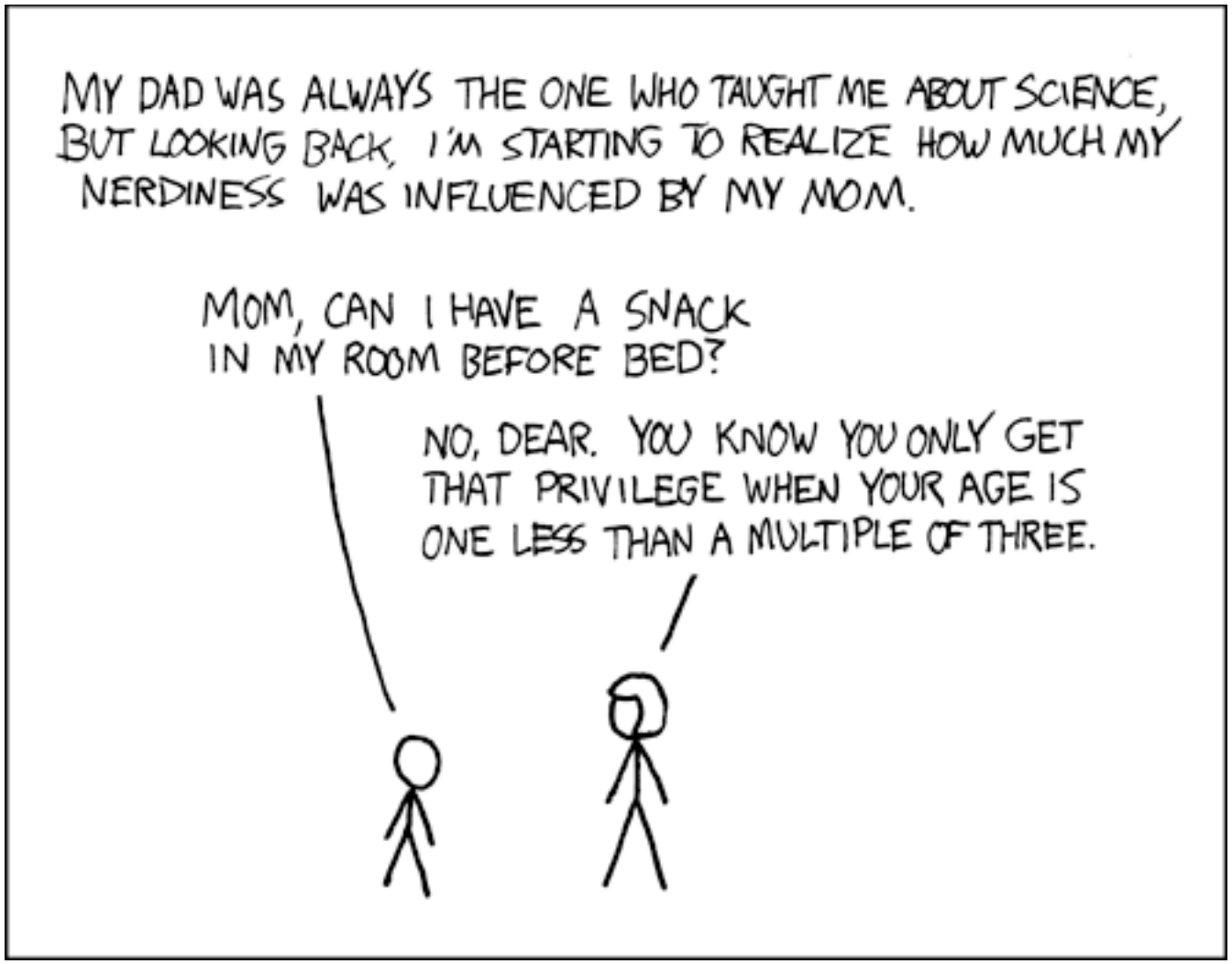
```
False                  // F

Var ("p")              // p

And(Var("p"),Or(Var("q"),(Var("r")))   // p ∧ (q ∨ r)
```

Specifying Basic Transformation Steps

- Rewrite rules
  - Substitution
  - Pattern Matching
  - Rule application
- Examples
  - Propositional formulae
  - Lambda calculus
  - Desugaring

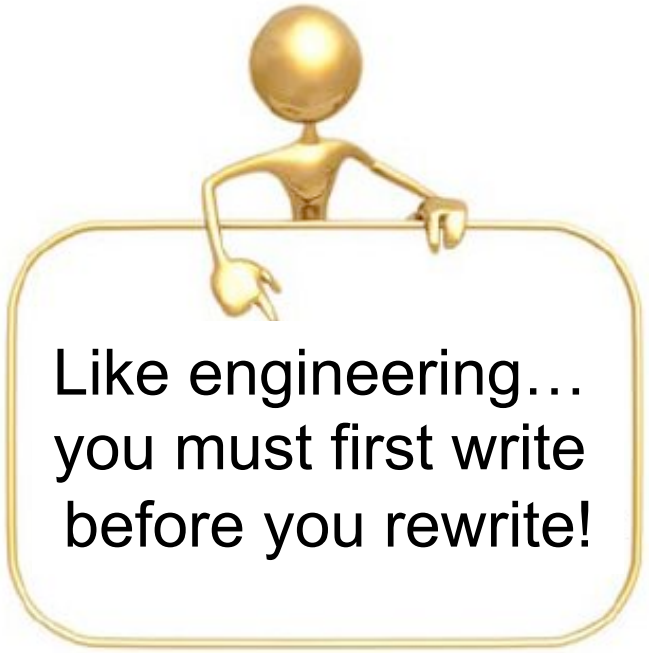


Let's talk about Rules...



Rewrite Rules

- Rule: `L : l → r`
  - Label/name `L`
  - Left-hand side pattern `l`
  - Right-hand side pattern `r`

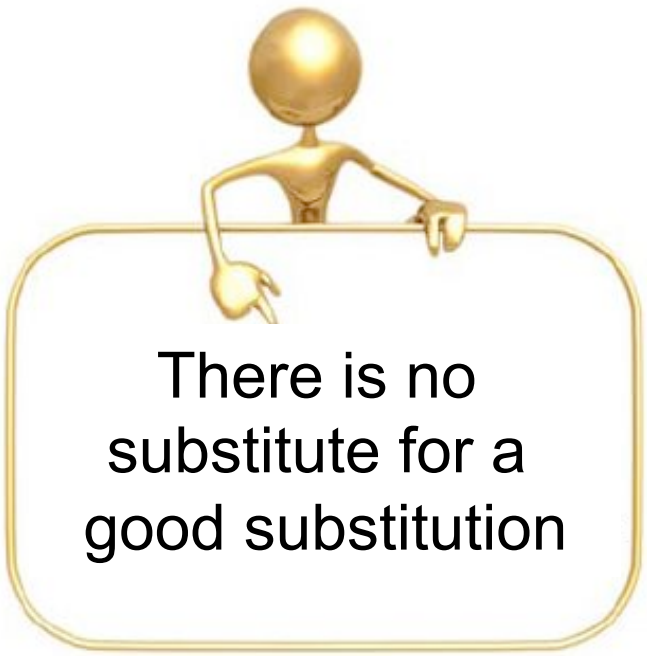


- Pattern: term with variables
  - `t := x | C(t1, …, tn) | C | int | string`
- Examples
  - `A : Plus(Zero, x) → x`
  - `B : Plus(Succ(x), y) → Succ(Plus(x, y))`

Substitution

- A *substitution* is a mapping from variables to terms

- Notation: `[t1/x1, … , tn/xn]` is a finite substitution mapping `xi` to `ti` and all other variables to themselves



- Application of a substitution `s` to a pattern
  - `subst(s, x) = s(x)`
  - `subst(s, str) = str`
  - `subst(s, num) = num`
  - `subst(s, C(t1, … , tn)) = C(subst(s,t1), … , subst(s,tn))`

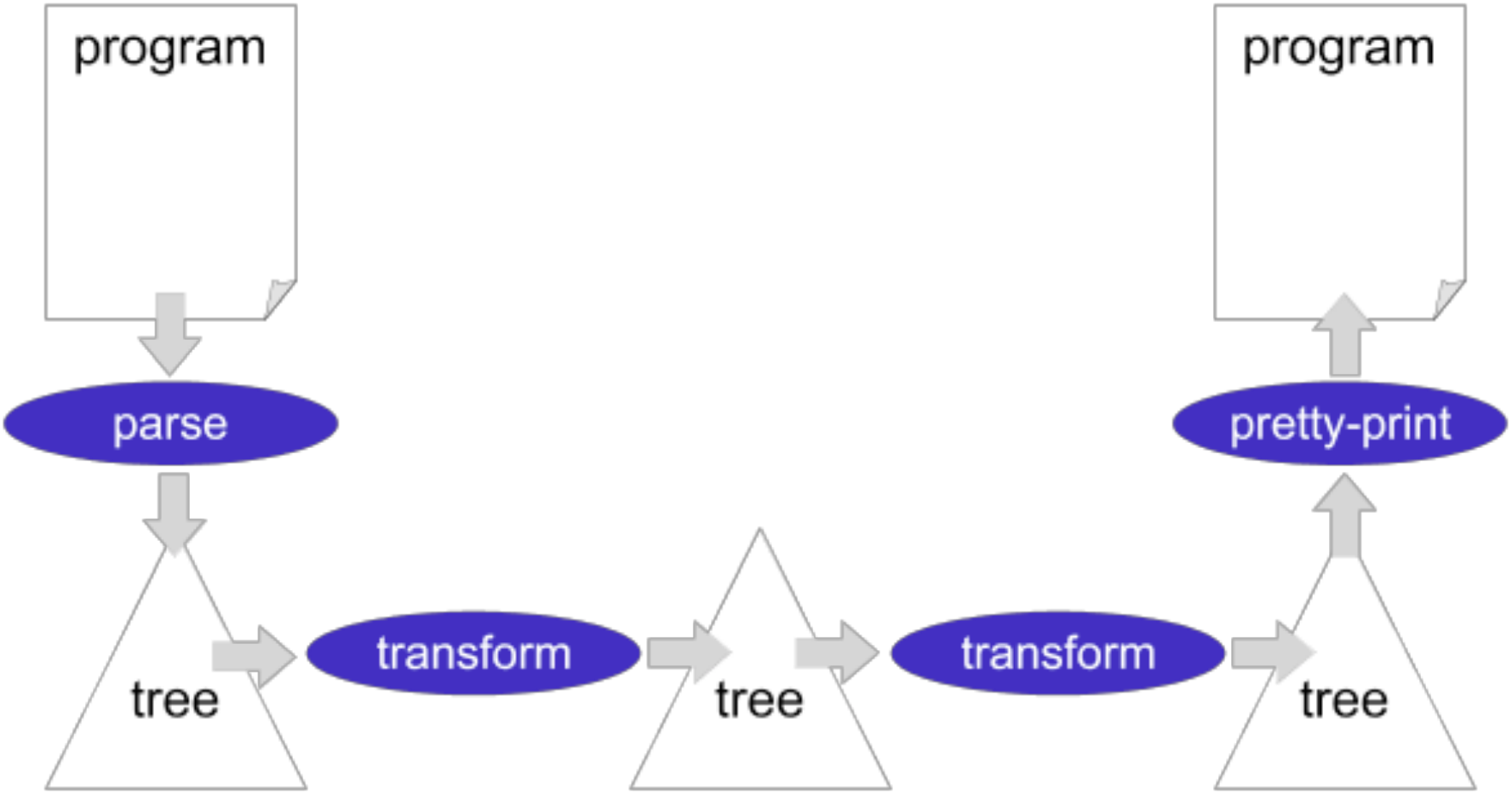
Term Pattern Matching

- A term `t` matches with a pattern `p` if there is a substitution `s` such that

  `subst(s,p) = t`

- Example
  - Pattern `Plus(Succ(x), y)`
  - Term `Plus(Succ(Zero), Plus(Succ(Zero),Zero))`
  - Substitution `[Zero/x, Plus(Succ(Zero),Zero)/y]`

Simple Transformational System

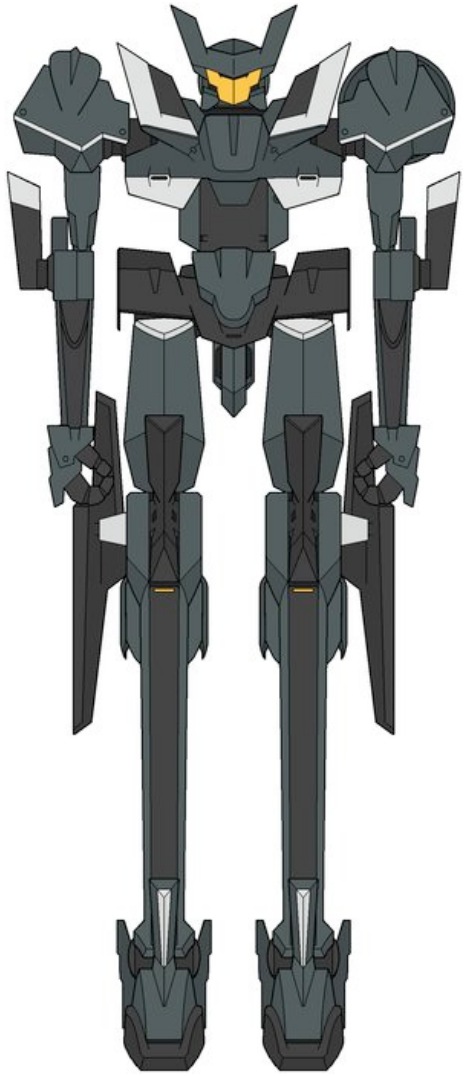


Example Rewrite Rules

```
InlineF :
 |[ let f(xs) = e in e'[f(es)] ]| ->
 |[ let f(xs) = e in e'[e[es/xs]] ]|

InlineV :
 |[ let x = e in e'[x] ]| -> |[ let x = e in e'[e] ]|

Dead :
 |[ let x = e in e' ]| -> |[ e' ]| where <not(in)> (x,e')


Extract(f,xs) :
 |[ e ]| -> |[ let f(xs) = e in f(xs) ]|


Hoist :
|[ let x = e1 in let f(xs) = e2 in e3 ]| ->
|[ let f(xs) = e2 in let x = e1 in e3 ]|
 where <not(in)> (x, <free-vars> e2)
```

Transformational Systems

- Correct programs can be built if the task is split into sufficiently small and formally justified steps
- Many of those steps are automatable
- If the automatable steps are performed by a machine, the programmer is free to focus on creative aspects of the job!



Transformational System Issues

- Specification vs. programming languages
- Level of automation – full, semi, user-driven
- Transformation mechanisms
  - Catalog approach:
    Production rules, knowledge-based systems
  - Generative set approach: Elementary transformations used in constructing new rules

Types of Transformational Systems

- Restructuring/Optimization
  - Same input and output language
- Conversion/Synthesis
  - Different input and output language

Transformational System Applications

- General support for program modification
- Program synthesis from a formal specification
- Adaptation to different environments
- Verification of program correctness

Applications of Software Transformation 1/2

- Compilers
  - Translation (e.g. Java into C#)
  - Desugaring (e.g. Java's foreach into for)
  - Instruction selection
    - Maximal munch vs BURG-style dynamic programming
  - Optimization
    - Data-flow optimization, Vectorization, GHC-style simplification, Deforestation, Domain-specific optimization, Partial evaluation…
  - Type checking
  - Specialization of dynamic typing

Applications of Software Transformation
2/2

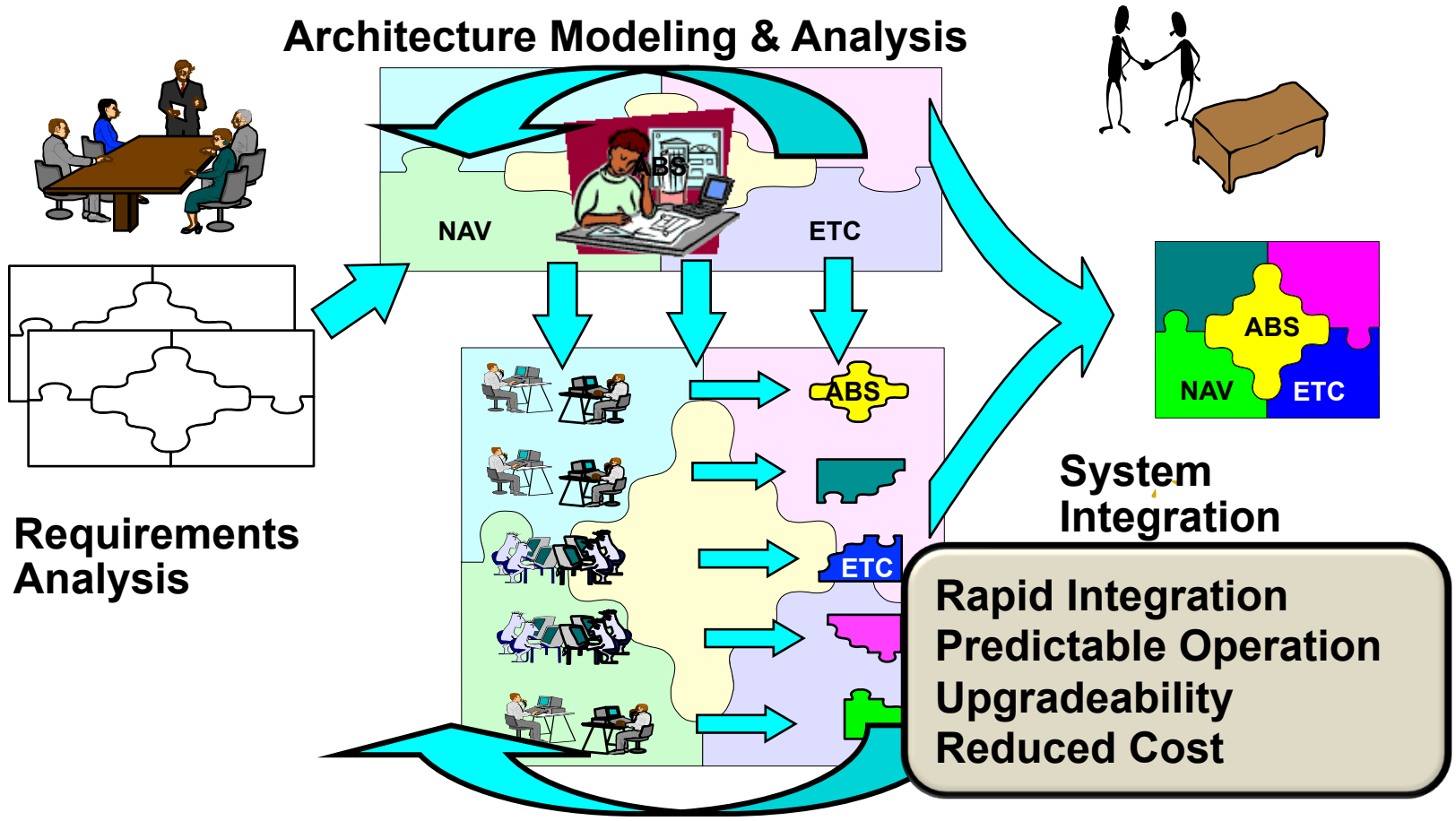
- Program generators
  - Pretty-printer and signature generation from syntax definitions
  - Application generation (e.g. data format checkers from specifications)
- Program migration
  - Platform conversion (e.g. MacOS to Linux)
- Program understanding
  - Documentation generation (e.g. JavaDoc)
- Document generation/transformation
  - Web/XML programming (server-side scripts)

So, What does this have to do with MBE?

- Reduces requirements errors as it forces rigor in the requirements analysis
  - Incompleteness and inconsistencies can be discovered and resolved
- Correctness by construction - preserving and guaranteeing essential properties
- Both specification and transformation rely on the rigors of Formal Specification and Transformation

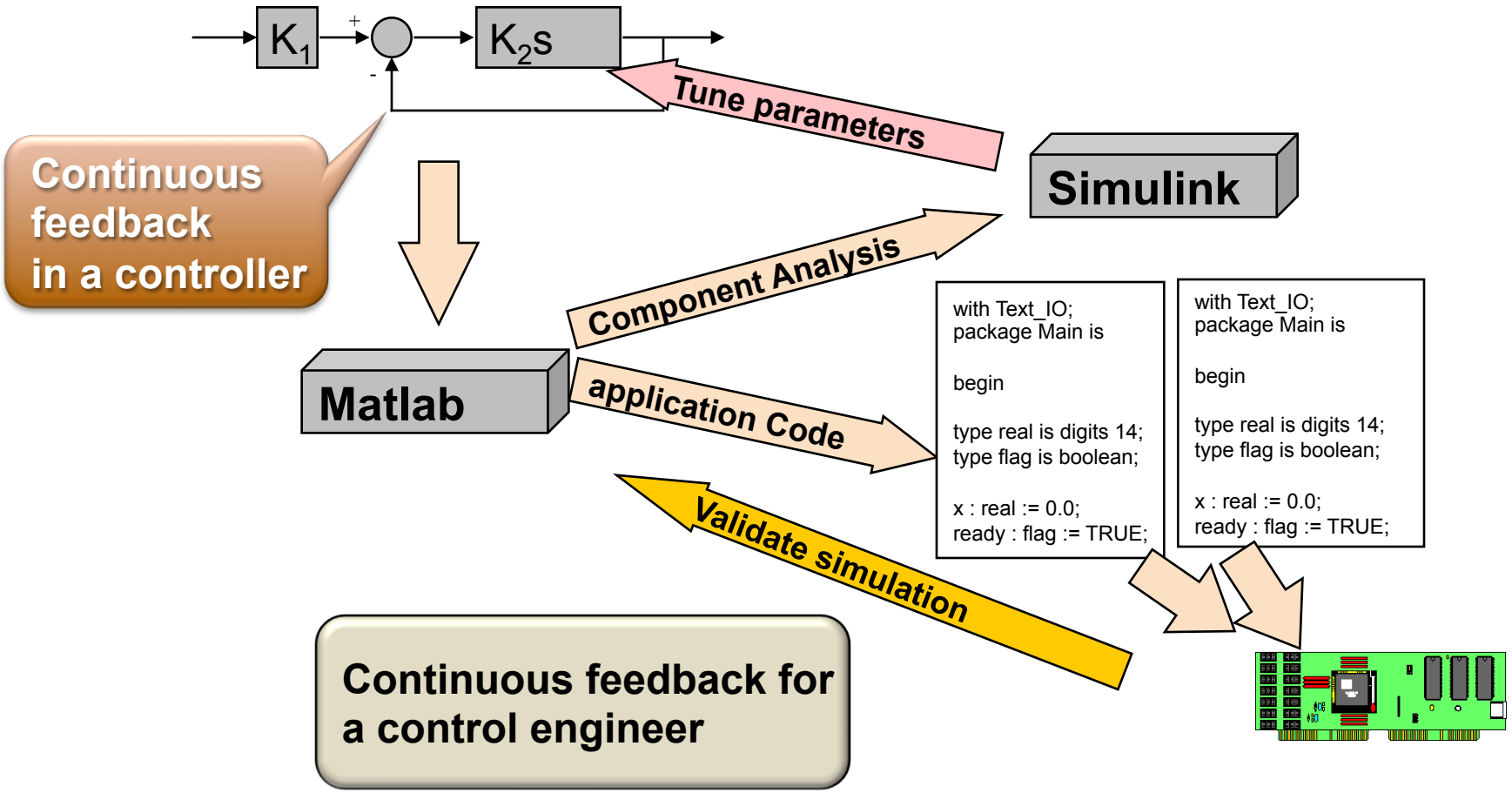
Model-Based System Engineering

(according to Software Engineering Institute)



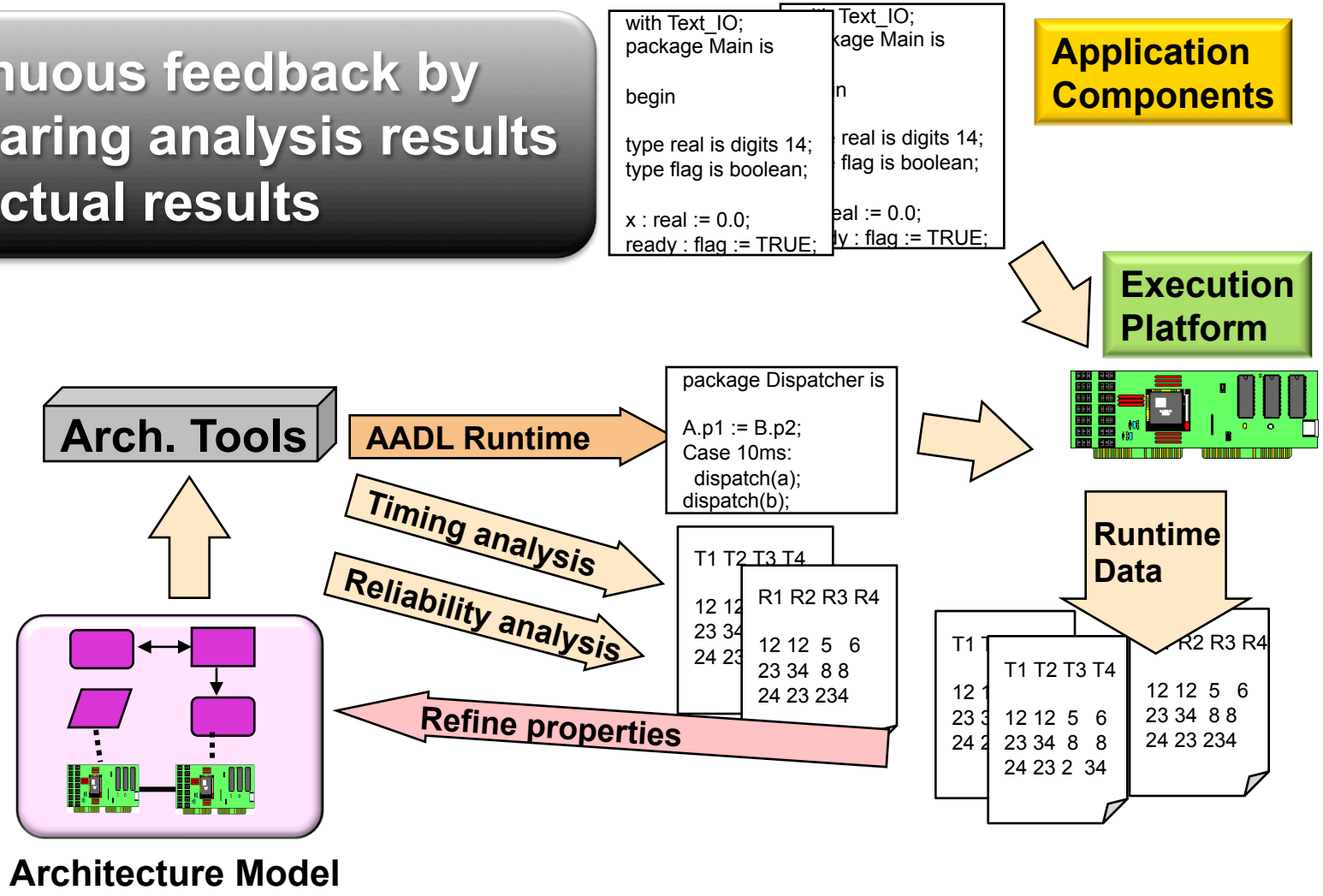
Predictive Analysis Early In & Throughout Life Cycle

A Control Engineer Perspective

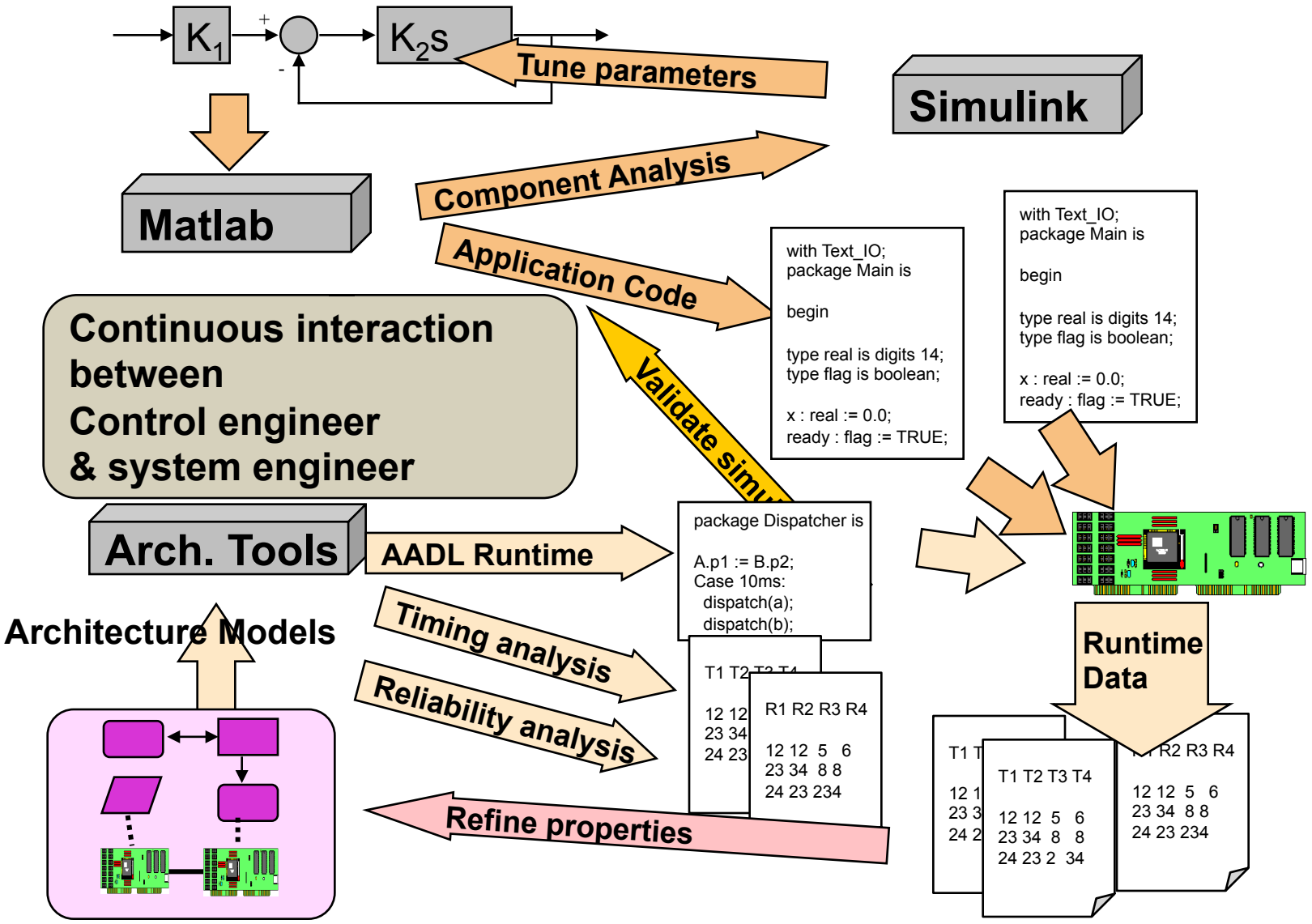


Software System Engineer Perspective

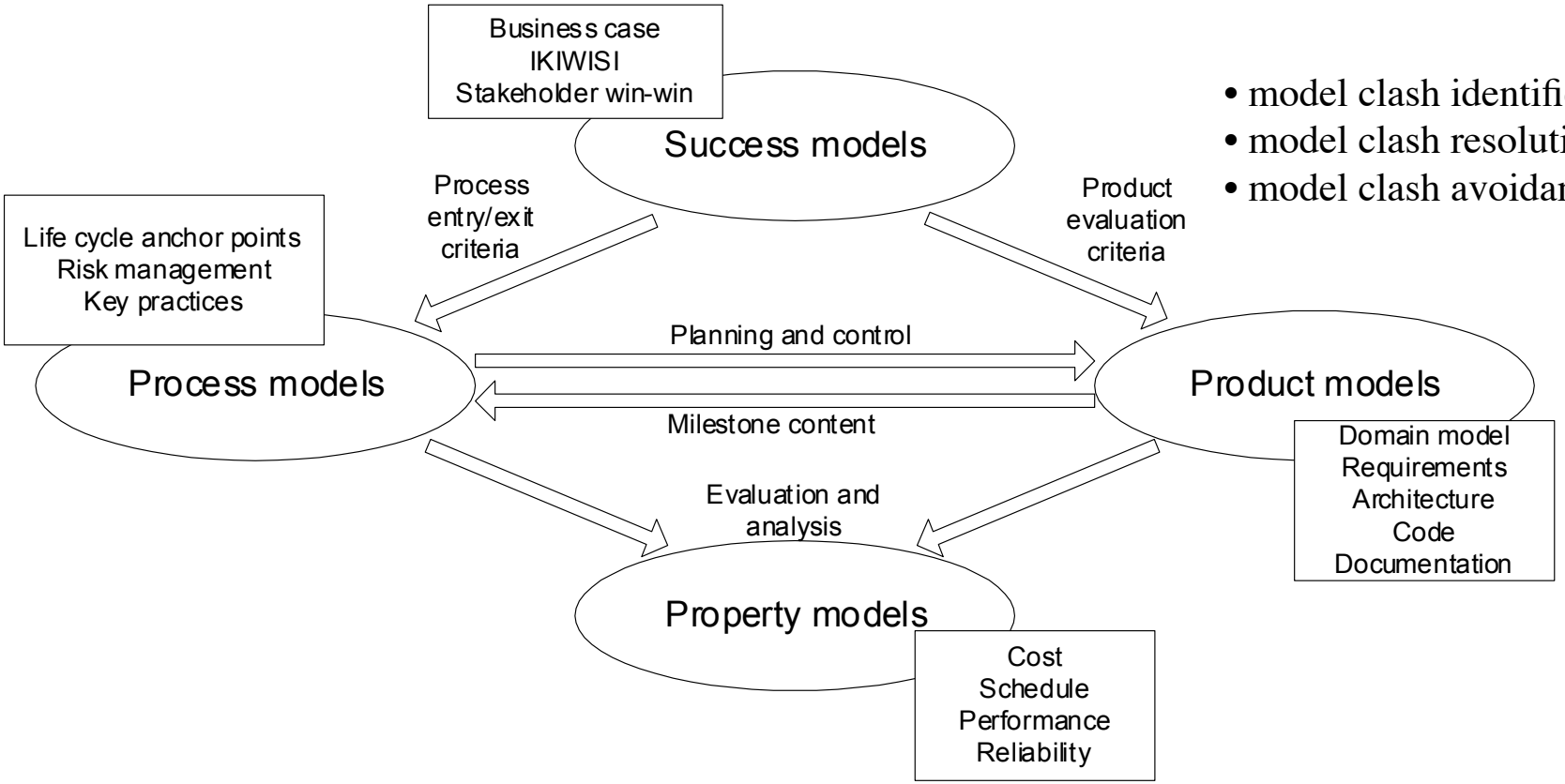
Continuous feedback by Comparing analysis results with actual results



A Combined Perspective



Multiperspective Model-Based Framework



- model clash identification
- model clash resolution
- model clash avoidance

Each perspective informs and provides evaluation criteria for the other perspectives.

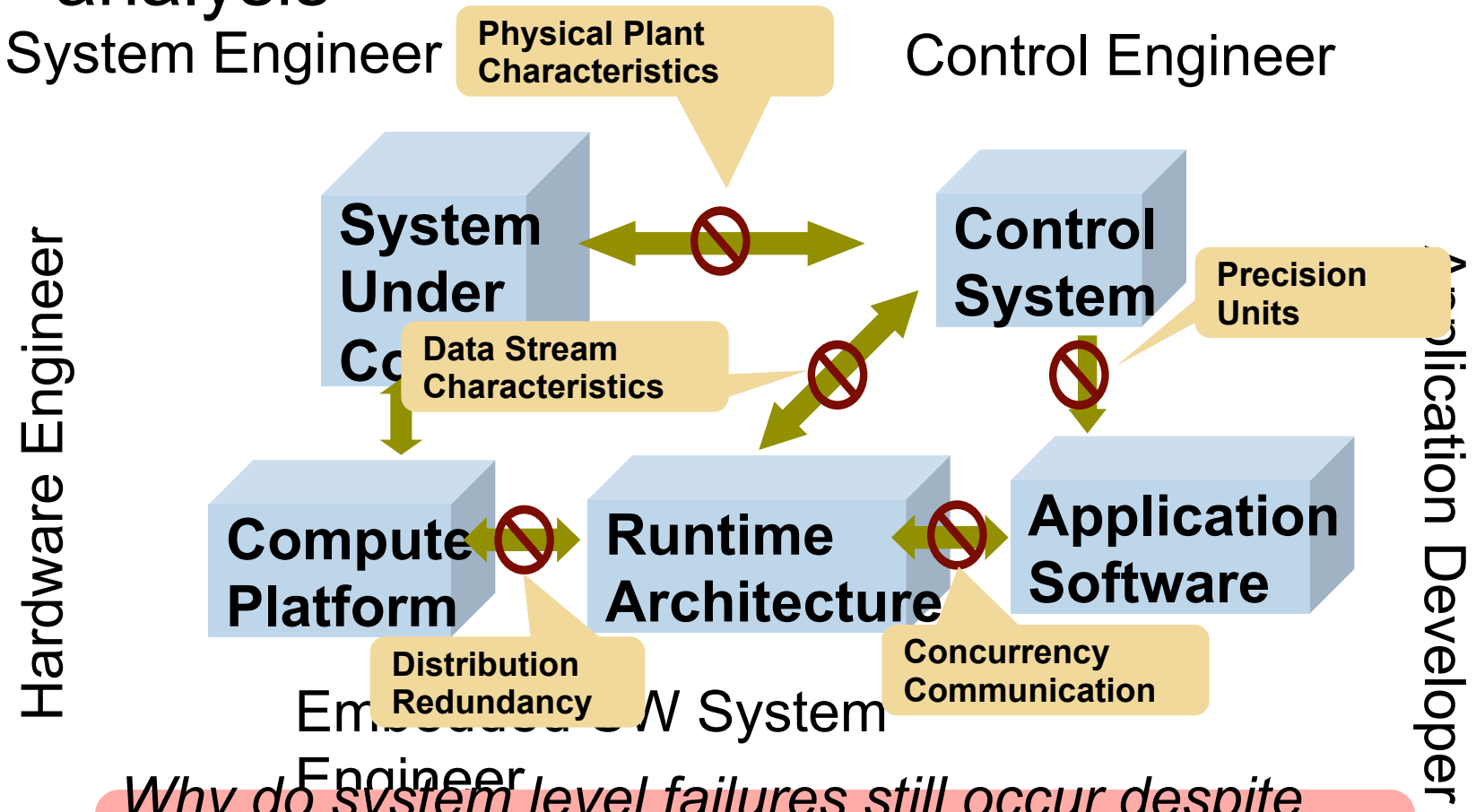
Late Discovery of System Problems

- **System integration problems**
  - System instability and failures
  - Implicit and mismatched assumptions
  - Shared computing resources
  - Complexity of component interaction
    - Functional
    - Extra-functional
- **Current practice**
  - Build components first
  - Then integrate and test
- **Way forward**
  - Analyze system models early and often
    (Virtual Integration)
  - Evolve components and integrated system

Mismatched Assumptions

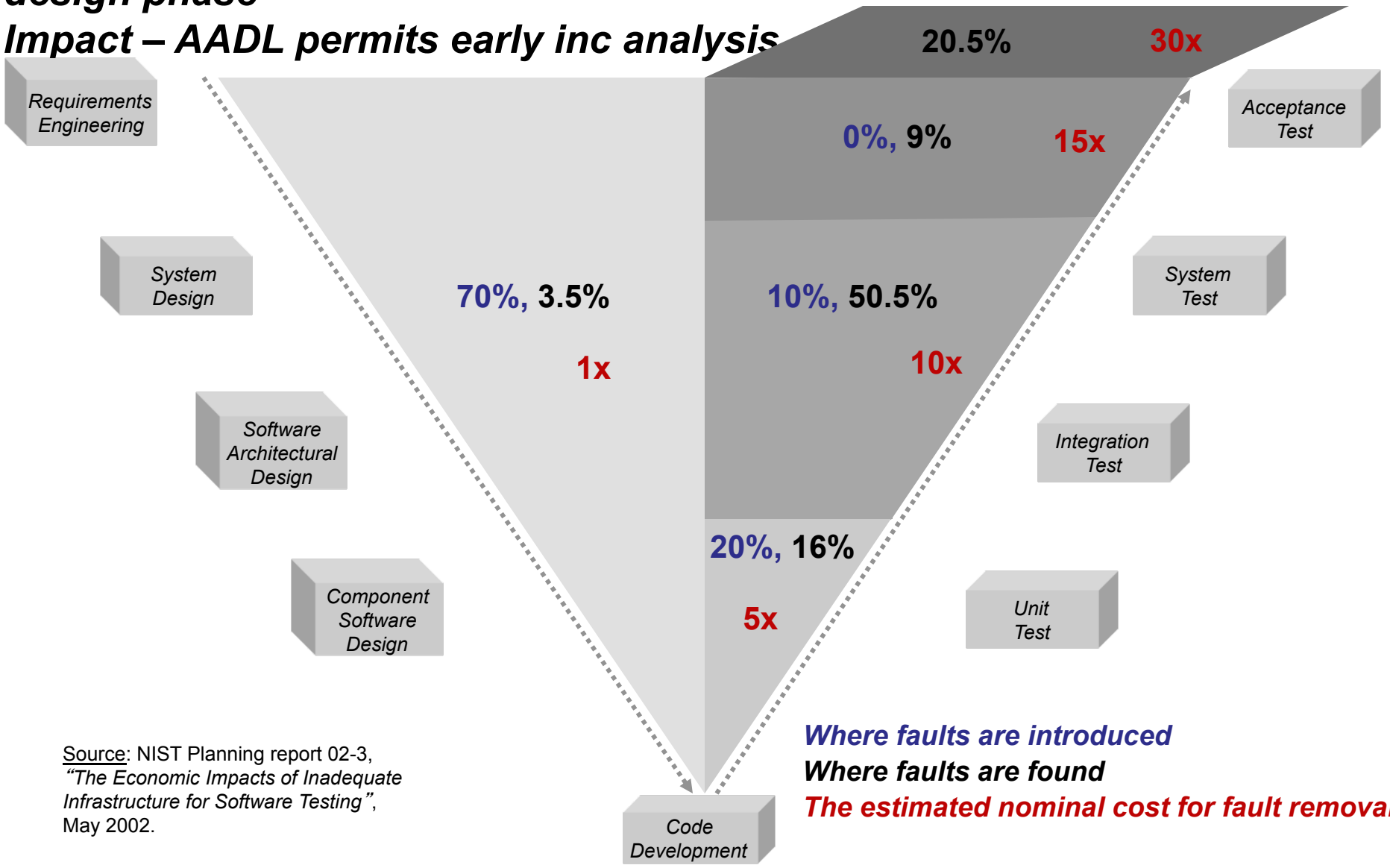
We Need Change

Impact – AADL integrates allowing analysis



Why do system level failures still occur despite fault tolerance techniques being deployed in systems?

MBE offers a way to find more faults in the requirements-architecture design phase
Impact – AADL permits early inc analysis



Source: NIST Planning report 02-3, *"The Economic Impacts of Inadequate Infrastructure for Software Testing"*, May 2002.

Where faults are introduced
Where faults are found
The estimated nominal cost for fault removal

Basic Model Layers

- One or model levels per layer
- Mappings and Transforms between models
  - Mappings
  - Transforms