Have we Learned from the Vasa Disaster?

Jean-Raymond Abrial

ETH Zurich

September 19th 2006

The Vasa Disaster

The Story

- August 10, 1628: The Swedish warship Vasa sank.

- This was her maiden voyage.

- She sailed about 1,300 meters only in Stockholm harbour.

- 53 lives were lost in the disaster.

Problems with the Vasa Construction

1. Changing requirements (by King Gustav II Adolf).

2. Lack of specifications (by Ship Builder Henrik Hybertsson).

3. Lack of explicit design (by Subcontractor Johan Isbrandsson)

   (No scientific calculation of the ship stability)

4. Test outcome was not followed (by Admiral Fleming)

References

- The Vasa: A Disaster Story with Software Analogies.
  By Linda Rising.
  The Software Practitioner, January-February 2001.

- Why the Vasa Sank: 10 Problems and Some Antidotes for Software Projects.
  By Richard E. Fairley and Mary Jane Willshire.
  IEEE Software, March-April 2003.

- The Vasa Museum
  http://www.vasamuseet.se

1. Requirements

The Classical Development Cycle

1. Feasibility Study

2. Requirement Analysis

3. Technical Specification

4. Design

4. Coding

5. Test

6. Documentation

7. Maintenance

An Analogy: a Mathematical Text Book

2.8 The Cantor-Bernstein Theorem.

If $a \preceq b$ *and* $b \preceq a$ *then* a *and* b *are equinumerous.*

This theorem was first conjectured by Cantor in 1895 and proved by Bernstein in 1898.

Proof: Since $b \preceq a$, then a has a subset c such that $b \approx c$.

...

$\square$

- In red: the reference text

- In blue: the explanatory text

Some Structuring Rules for the Requirement Document

- Two separate texts in the same document:
  - explanatory text: the why
  - reference text: the what

- Embedding the reference text within the explanatory text

- The reference text eventually becomes the official document

- Must be signed by concerned parties

The Reference Text

- Contains the definition and properties of the future system

- Made of short labeled English fragments (traceability)

- Should be easy to read (different font) and easy to extract

- About the abstraction levels (don’t care too much)

- The problem of over-specification (don’t care too much)

2. Specifications and Design

What Engineers Should Do for Specifying and Designing?

- Engineers should construct models of the intended system

- Thus, execution is not possible (at least initially)

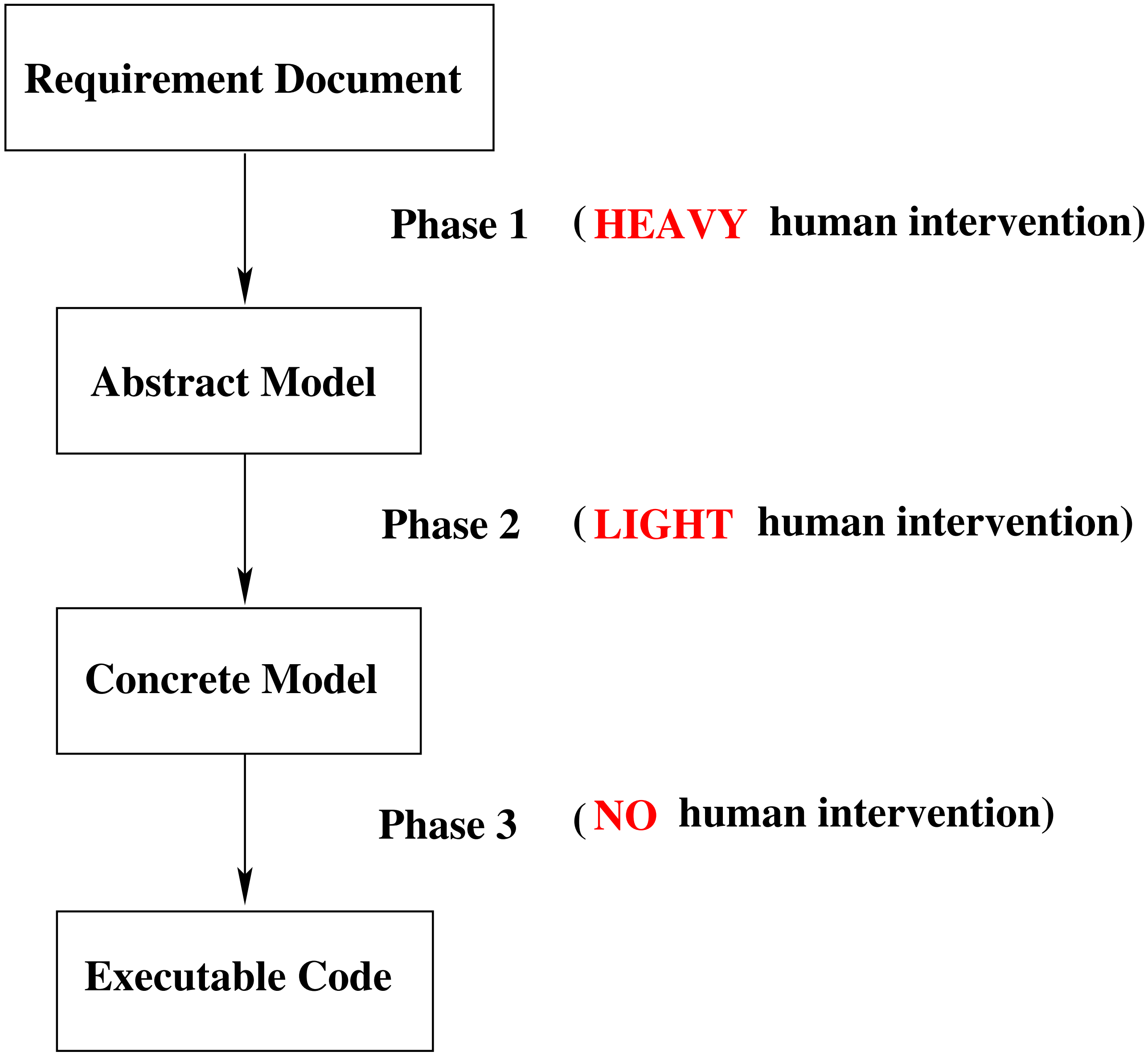
- But engineers will still make mistakes

- How can such mistakes be discovered (if no execution)?

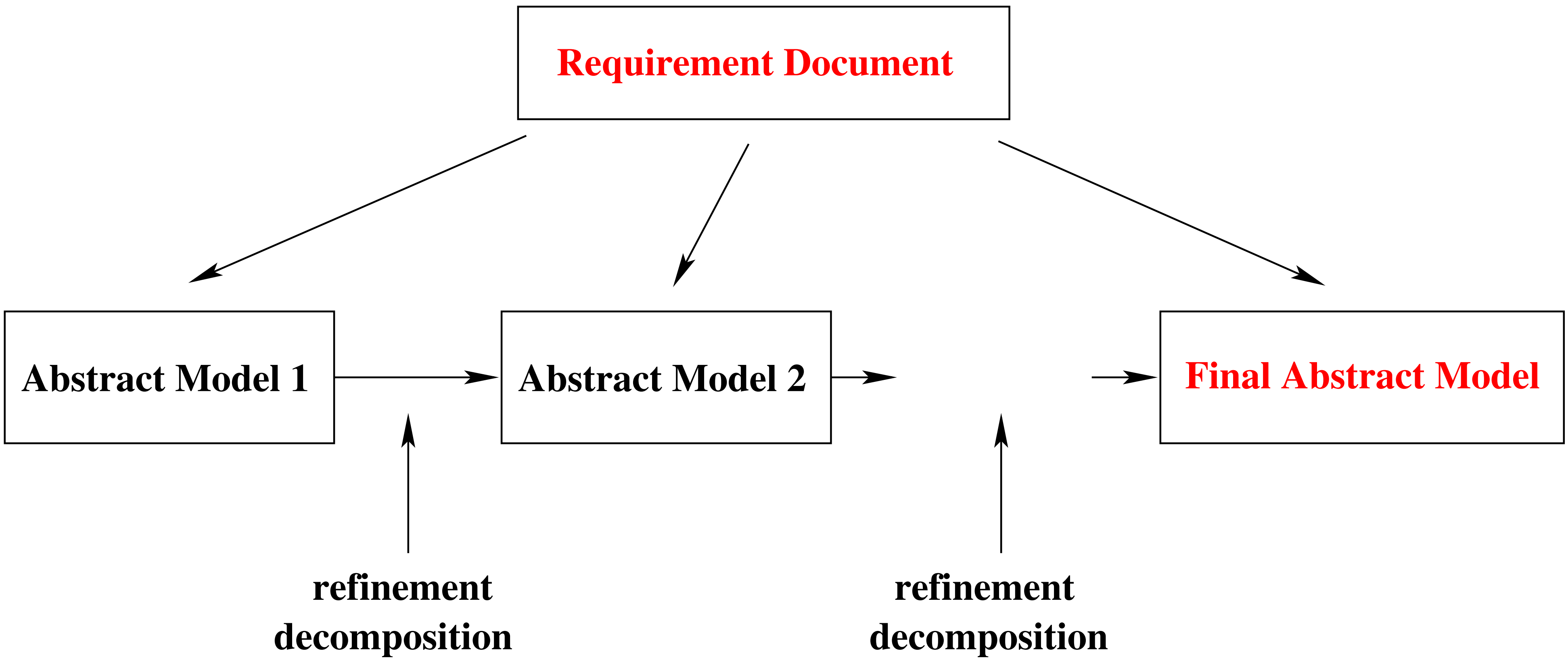
- Answer: by doing proofs

- The goal: to have systems being CORRECT BY CONSTRUCTION

Three Distinct Phases in Formal Development

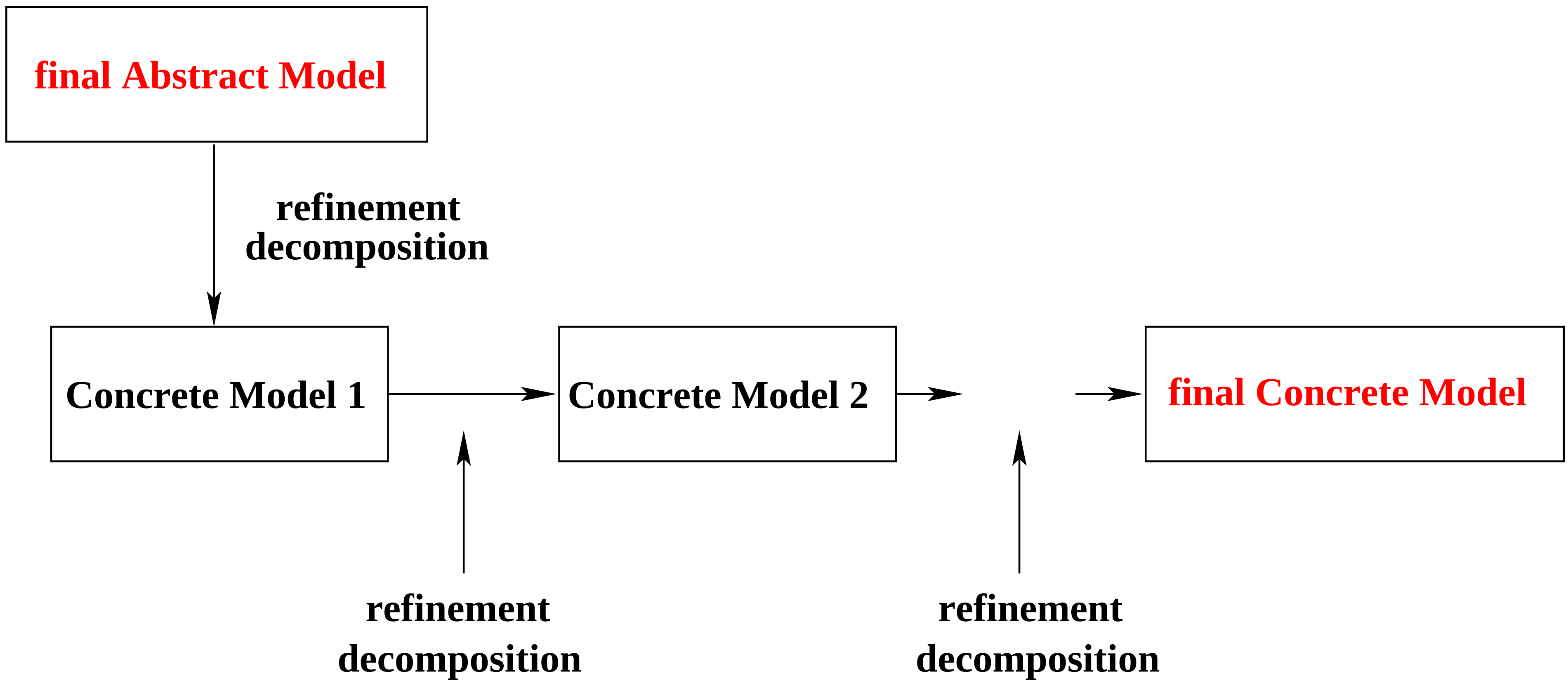


Phase 1: Constructing the Abstract Model



- Superposition Refinement

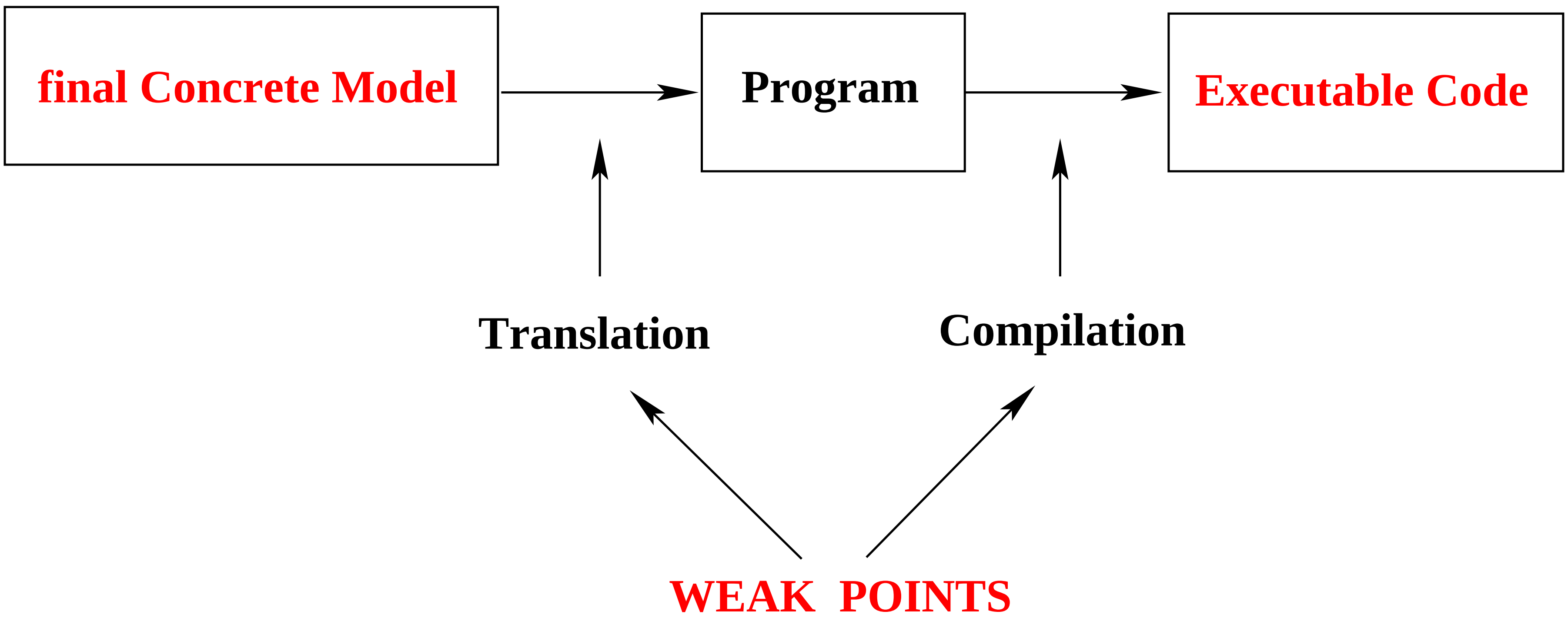
Phase 2: Constructing the Concrete Model



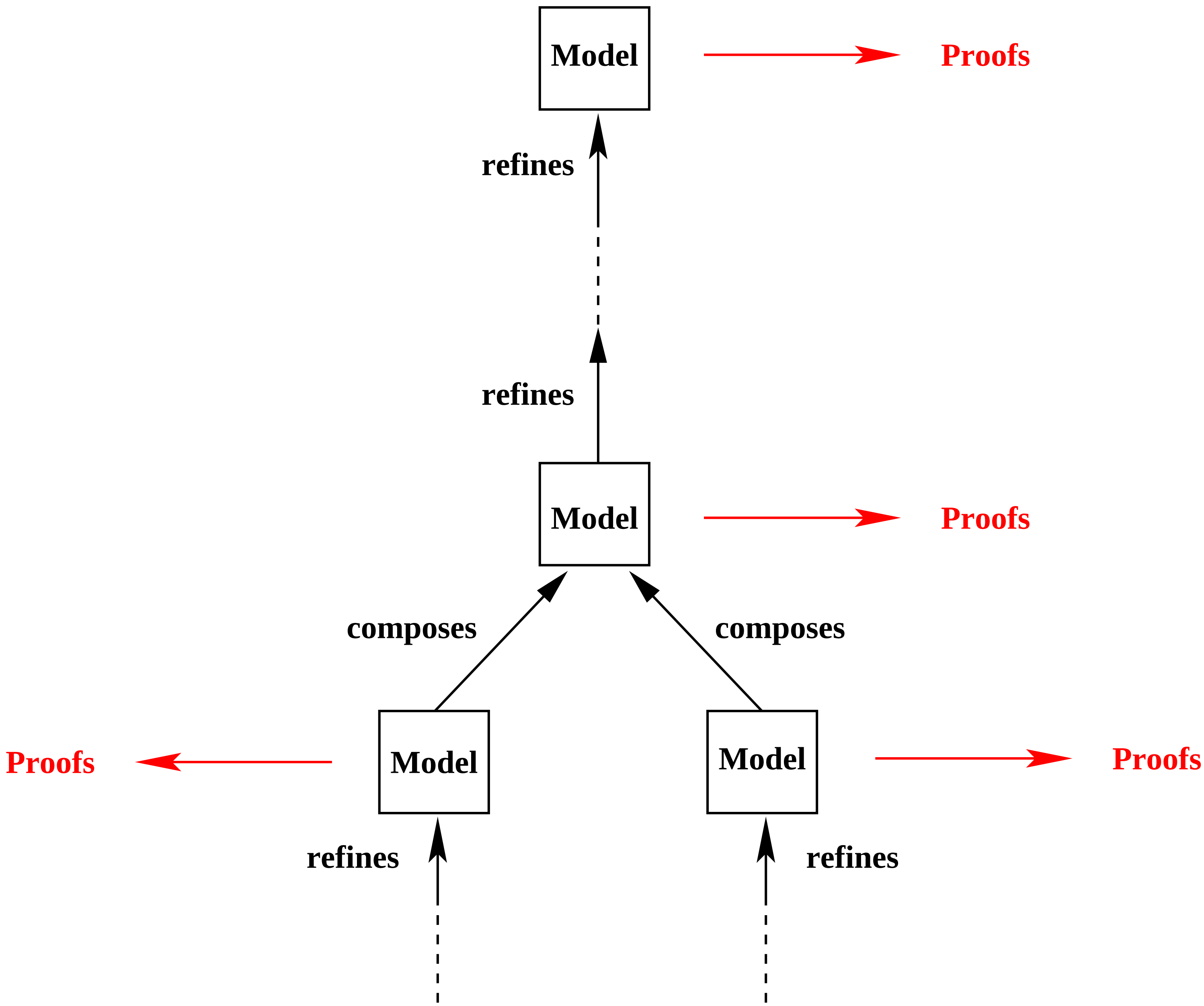
- Data and algorithmic refinement

- Can be partially done by a refining tool

Phase 3: Getting the Executable Code

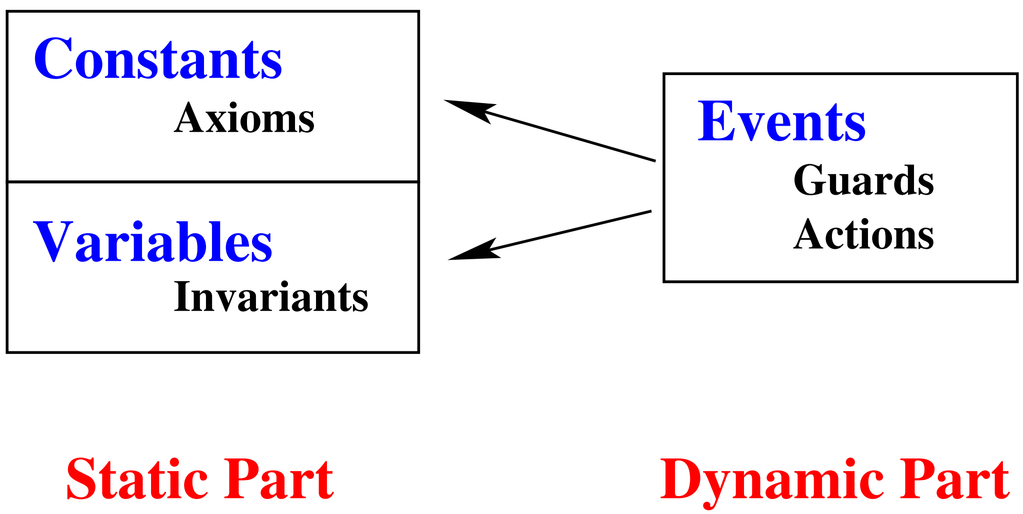


The Modelling Process



Modelling Elements (Constituents of a Model)

- Axioms, invariants , guards, and actions are written using the notation of first order logic and that of set theory



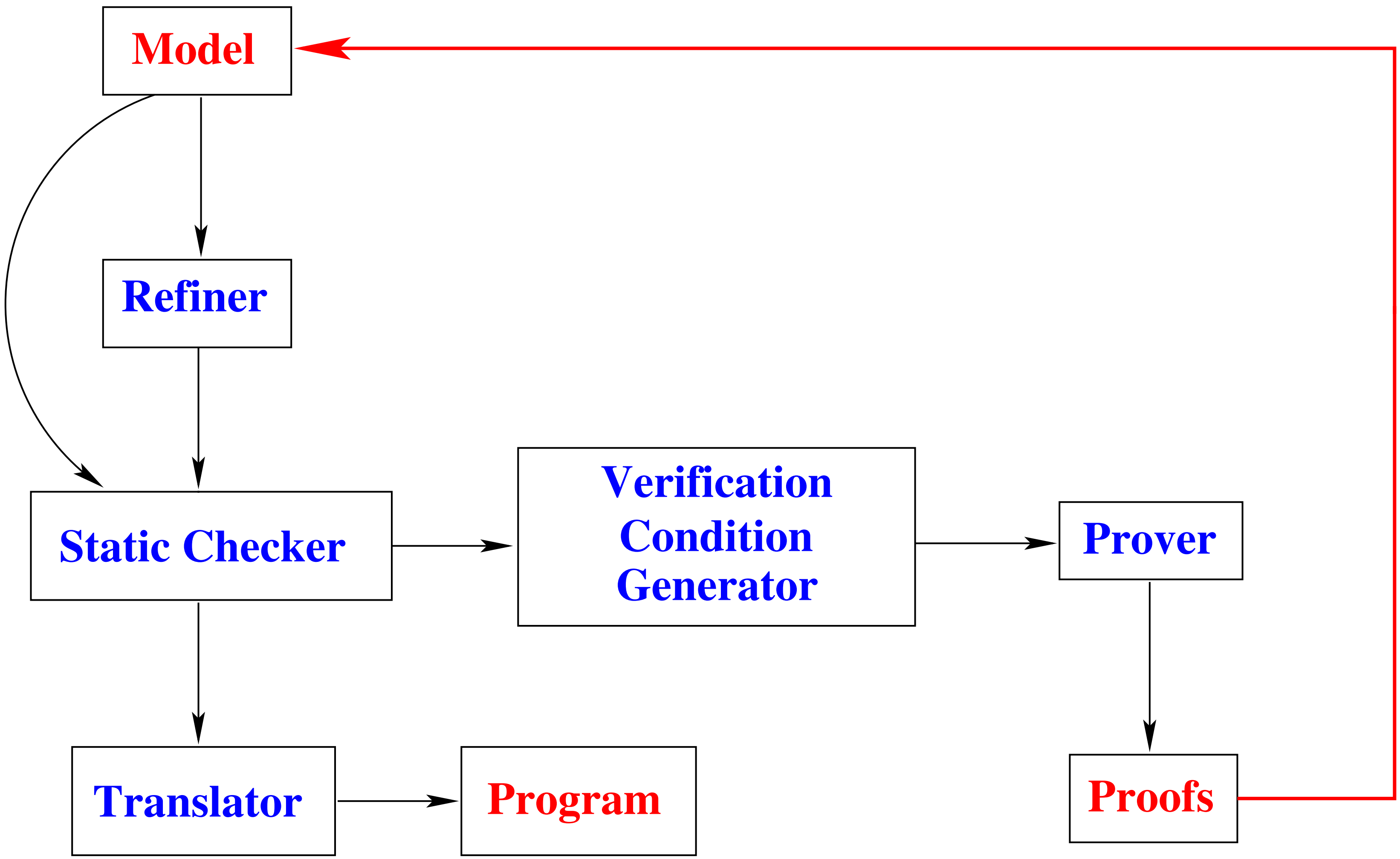
- Reference: R. Back and R. Kurki-Suonio, Distributed Co-operation with Action Systems. ACM Transactions on Programming Languages and Systems. October 1988

Verification Conditions

Some formal verification conditions can be used to prove:

- correct invariant preservation
- correct refinement
- correct new event additions in a refinement
- correct decomposition
- possible deadlock freedom

Summary of Tools



3. Is it at all possible?

Difficulties

- Difficulties with the requirement document

- Difficulties in constructing models

- Difficulties with proving

- Other difficulties

Difficulties with the Requirement Document

- Important because it is the point of departure of the development

- Errors or omissions in this doc. might remain in the development

- A formal approach does not guarantee to discover these problems

- Although proofs help discovering inconsistencies

- UML is not the solution

- Suggestion: Systematic re-writing of this document

Difficulties in Constructing Models

- Modeling is a difficult task

- The order in which to extract requirements is not obvious

- Software engineers are usually not well educated in modeling

- [Nor are they for requirement document writing]

- The gradual construction of models is not mastered

- People tend to make too few refinement steps

Instructing Software Engineers

- Engineers have no problem to learn the mathematical notations

- They have more difficulties to master the construction of models

- The following disciplines have to be developed in CS curriculum:
    - Requirement document writing
    - Model construction

- This is what I am trying to do at ETH in Zurich

About Proving

- Proving is not a difficulty

- Properties to be proved are determined a priori

- They are part of the model

- They are not chosen a posteriori as in testing

- Modeling versus programming: an important distinction

- Modeling allows us to reason about our intended system

Analyzing the Results of Proving: Debugging the Model

- Proof succeeds: our ultimate goal

- Proof provides a counter-example: model has to be modified

- Proof fails but is probably feasible: model has to be reorganized

- Proof fails and is probably not feasible: model has to be enriched

- Proving is not a goal per se

- It is an excellent basis for asking questions

Another Difficulty

- Integration of approach within the development process

- This is probably the most serious obstacle

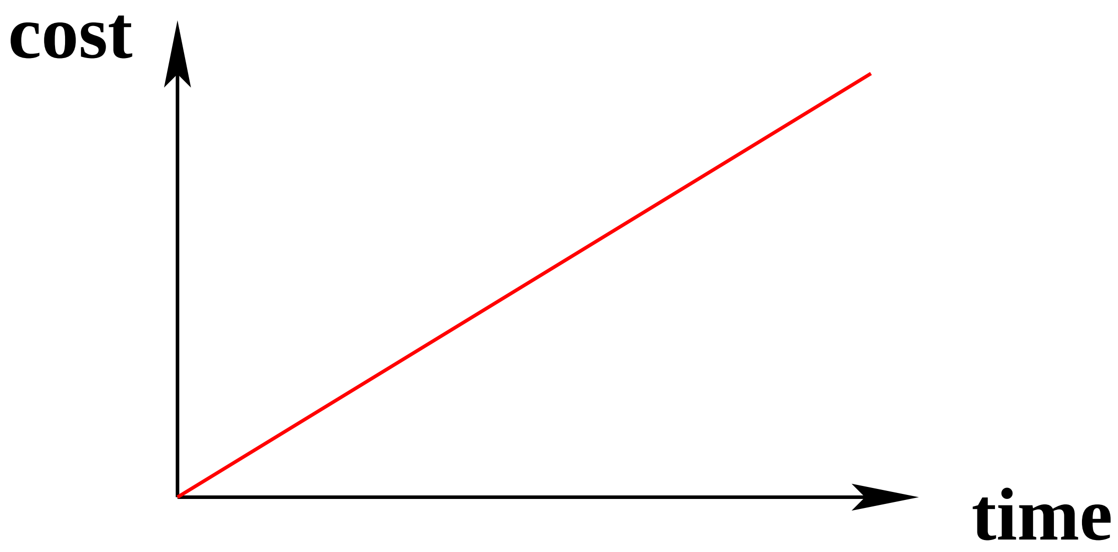
- Such processes are difficult to define and then to put in place

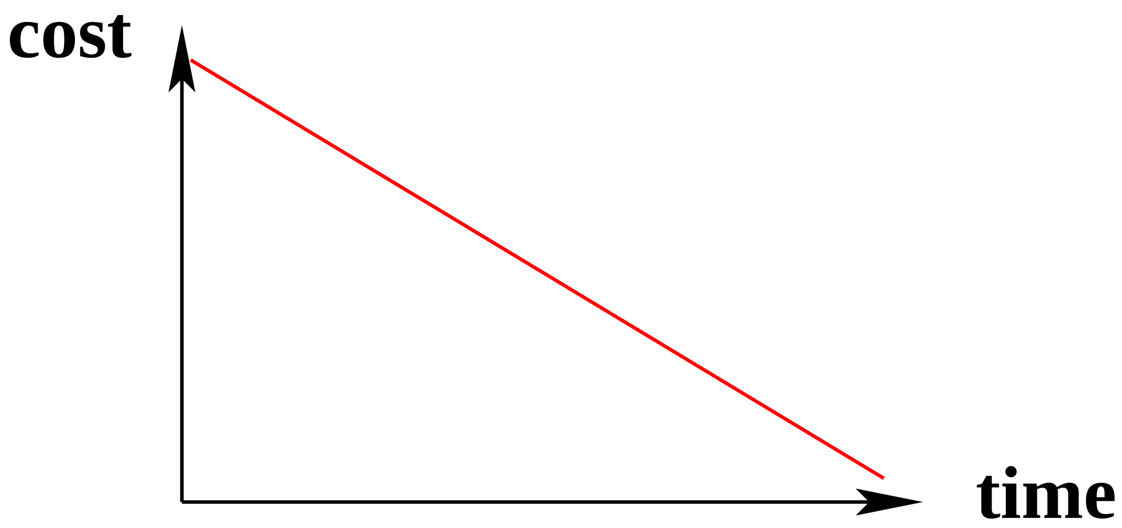
- Thus managers are reluctant to modify them

- Early phases are more costly than in more classical development

- Final phases (coding, integration, testing) are far less costly

Managers don't Trust the Second Curve





4. Preferred candidates for this approach: embedded systems

Embedded Systems (1)

- It is to be opposed to a general purpose computer system like a PC Operating System

- The computer is encapsulated within the device it controls

- It is doing for ever a number of specific tasks

- Examples: Systems controlling
  - a portable telephone
  - an aircraft or a space ship
  - a driverless train
  - a nuclear reactor
  - ...

Embedded Systems (2)

- Such systems are working in close connection with an external often unpredictable environment (physical and human)

- Reliability is usually very important

- Error detection and recovery must be performed (degraded mode)

- Real-time constraints have to be taken into account

- Consequently, the software has to be developed with great care

5. Some Conclusion

Less Usage of Programming Languages

- I am convinced that Programming Languages (and OO) will be less used in the future for constructing embedded systems

- The classical notion of source file will disappear

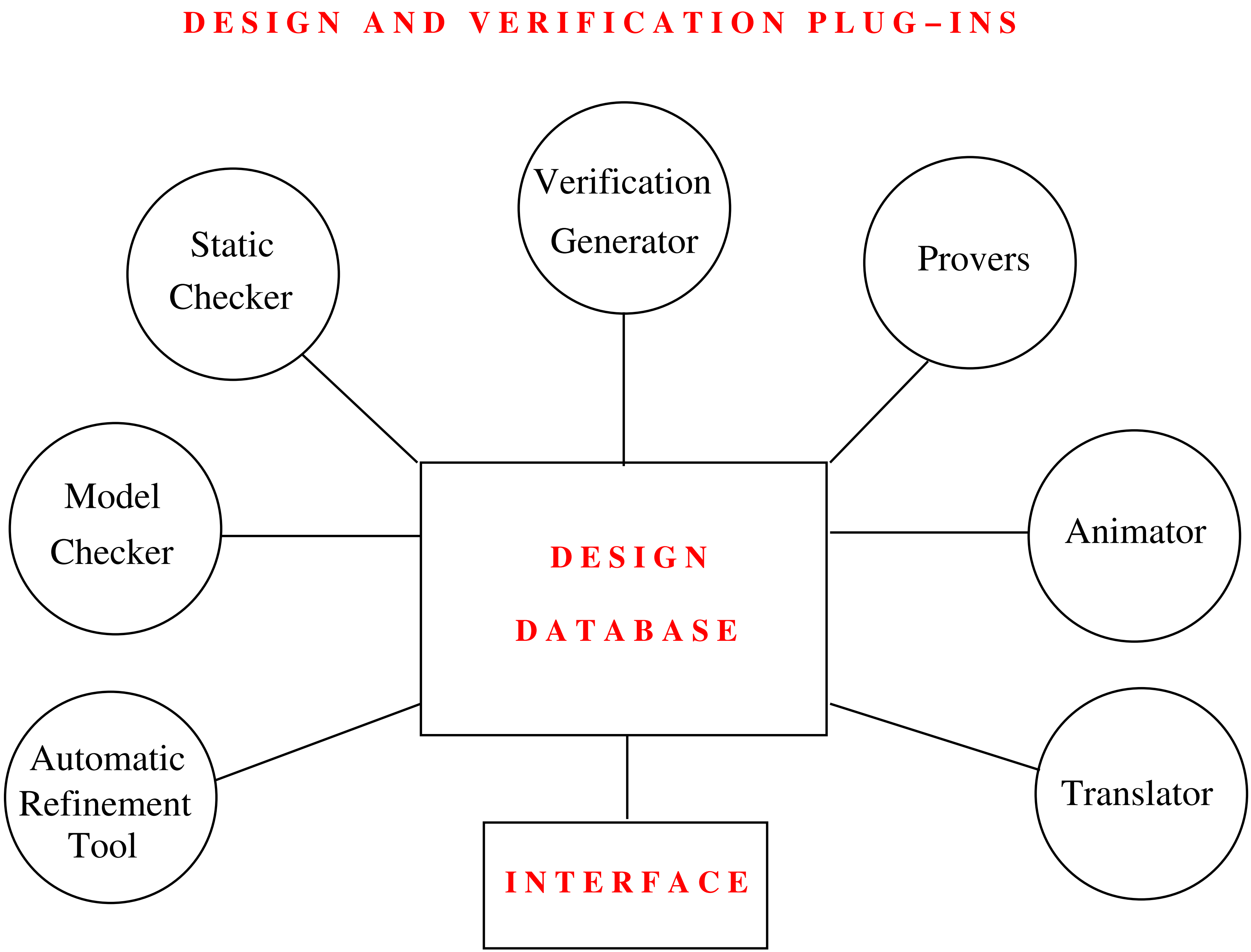
- It will be replaced by a specification and design database

- Code will be generated automatically

- This tendency is already there: Eclipse

- This is what we do in the Rodin EU Project: http://rodin.cs.ncl.ac.uk

The "Compiler" of Tomorrow



Thanks for Listening