EDOM - Engenharia de Domínio

Mestrado em Engenharia Informática

Lecture 01.1

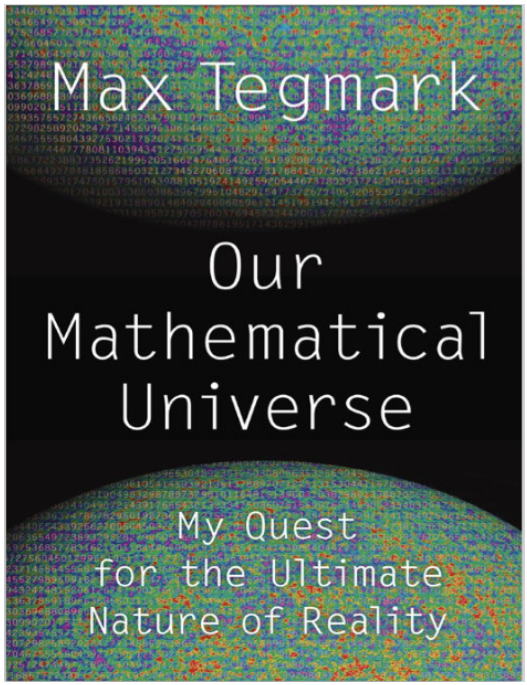
EDOM Introduction

Alexandre Bragança atb@isep.ipp.pt

Dep. de Engenharia Informática – ISEP

2017/2018

Side Thought: What is Reality?



"Our Mathematical Universe", Max Tegmark, Alfred A. Knopf, 2014
Max Erik Tegmark is a cosmologist. Tegmark is a professor at the Massachusetts Institute of Technology

- This book is about reality.
- What is our reality? What is our universe? How it began? How does it function?
- The author explores ideas such as infinity and parallel universes.
- According to the author there are 4 levels of parallel universes, each level is constrained by the next level.
- This is a book of Theoretical physics, a branch of physics which employs mathematical **models and abstractions** of physical objects and systems to rationalize, explain and predict natural phenomena.
- This course is essentially about **models and metamodels** and how to use them in software engineering.

Context

- This course is mandatory for the software engineering branch of the program.
- Domain engineering has gain a lot of attention from the academia and the industry because of recent scientific and practice innovations originating from specific approaches that are related to it, namely, **model-driven engineering**, **domain-specific languages** and **product line engineering**.
- The course will address these specific topics in the context of the domain engineering discipline.
- A project will be developed using a Project-Based Learning (PBL) approach that will use the GWT framework.

So long...

Say hello to...

Goal

- The major goal of this course is to provide students with **advanced skills regarding domain engineering** and some of the most recent scientific innovations and practices originating from specific approaches that are related to it: model-driven engineering, domain-specific languages and product line engineering.

Outcomes

By the end of this course the student will be able to:

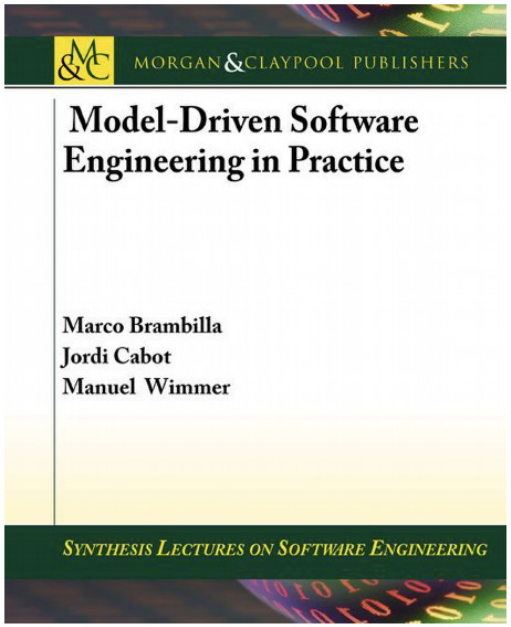
- Analyze domain engineering as the top of an iceberg composed of a myriad of processes, techniques, tools and methodologies which all have reuse in software engineering as the ultimate goal;
- Analyze software product lines as a practice that promotes large scale reuse;
- Design domain-specific languages in the context of model-driven engineering;
- Manage recent approaches in the context of domain engineering and reuse: model-driven software engineering and domain-specific languages
- Debate about options in projects related to domain engineering

Contents (1/2)

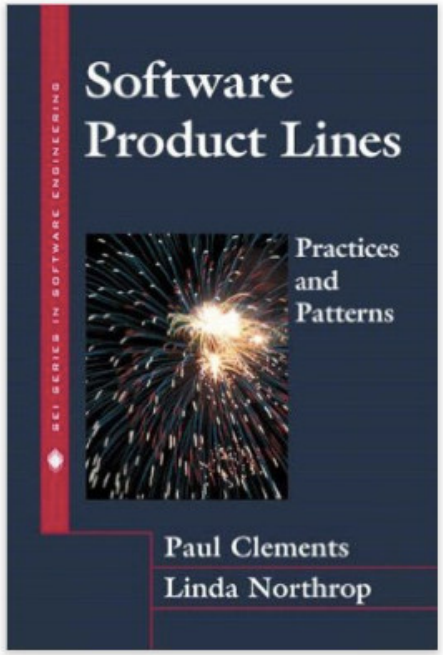
Contents (2/2)

- Domain-specific Languages
  - The Eclipse Modeling Framework (EMF)
  - Developing a modeling language
  - Abstract syntax and concrete syntax
  - Model to Model transformations
  - Out-place transformations with Atlas Transformation Language (ATL)
  - In-place transformations with Graphs
  - Model to Text transformations
  - Code generation.
  - Managing Models
- Software Product lines (SPL)
  - Core activities: domain engineering, application engineering and product line management
  - SPL practice patterns
  - Variability Models

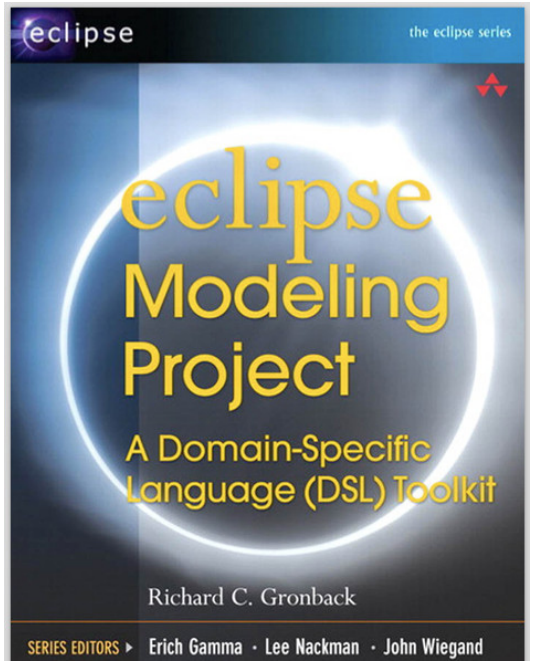
Bibliography



"Model-Driven Software Engineering in Practice, 2nd Edition", Marco Brambilla et al., Morgan & Claypool Publishers, 2017



"Software Product Lines: Practices and Patterns", Paul Clements, Linda Northrop, Addison-Wesley, 2001



"Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit", Richard C. Gronback, Addison-Wesley, 2009

Teaching/Learning Tools

- Moodle (`moodle.isep.ipp.pt`)
- Eclipse. Available at `http://eclipse.org`

Previous knowledge assumed as acquired

- Extended knowledge and experience in programming in the java language (LEI: APROG, PPROG, LPROG).
- Software engineering knowledge and experience, mainly analysis and design best practices with object oriented paradigm and UML (LEI: ESOFT, EAPLI).
- Basic knowledge and experience with version control systems (LEI: LAPR2, LAPR3, LAPR4 and LAPR5).

Classes

Lectures

- Lectures will usually be expositive, when introducing or explaining concepts. Lectures will also have demonstrations when the topic is connected with the lab classes.

Lab Classes

- Lab classes usually operate as hands-on, where students solve small exercises (first weeks).
- After these first small exercises, students work on a project.

Tutorial Classes

- Used to support the exercises, project and other issues. Students may propose a topic or issue to be discussed in the class.

Assessment

Assessment during the semester (Exercises [25%] + Project [45%]) and final mandatory exam (30%).

- Students must form teams of 3 elements to work during the semester.
- Each group will work within a specific repository.
- All artifacts produced during the assignments and project are required to be committed to the repository.
- Students who are free from attending classes must develop the assignments and project outside classes. They must comply with all the other rules (e.g., deadlines).

Classroom Assignments (i.e, exercises) [25%]

- Set of exercises. No minimum grade.

Project [45%]

- Project to be developed in 2 iterations. Minimum score of 10,0/20,0.

Exam [30%]

- The final exam has a minimum score of 8,0/20,0.

Assessment - Classroom Assignment [25%]/ Presentation

Formative and Summative

Set of small exercises to be executed during the lab classes.

- Each exercise includes 2 components: the development of one **base solution** and one **alternative solution**[1].
- Components are developed individually.
- Each student must receive individual feedback on his performance on 3 components.
- This individual feedback is a major input for the grading of this assessment tool. The student chooses the components to be assessed but must include components of all types.

Feedback

- Feedback of every assignment requires the presentation and debate of every student (it can be made by video-conference in accordance with the evaluator teacher), or a performance of 0 (zero) will be assigned to the student for the exercise.
- Students are allowed to improve one exercise but are required to inform the teacher in the same week.

[1] An alternative is a solution that satisfies the same requirements but with a different process.

Assessment - Classroom Assignment [25%]/ Grading Criteria

The feedback rating scale for each exercise is from 0 to 4, with the following semantics:

- **0- nothing**. No submission.
- **1- tentative**. Did not achieve the requirements of the exercise.
- **2- acceptable**. Achieved, partially, the requirements of the exercise
- **3- good**. Achieved, completely, the requirements of the exercise (includes incomplete analysis).
- **4- very good**. Achieved, completely, the requirements of the exercise, with outstanding performance in the assessed outcomes (includes rigorous analysis).

Notes:

- Levels 3 and 4 can only be achieved if the exercise includes analysis.
- Analysis includes a justification of options and also a comparison with the other solution.
- The final score is based on all the qualitative performances of the student in the assessed exercises as well as the teacher assessment of the global performance of the student in accordance with the specific course outcomes addressed by CA and also organization and team work.

Assessment - Project [45%]/ Presentation

Formative and Summative

Project to be developed regarding model-driven engineering. The project has 2 iterations/deadlines.

- Feedback is given to students in the first deadline.
- The weight of the iterations is: 25% for the first and 75% for the second.
- There will be a document with the project statement.
- The project has a minimum score of 10,0/20,0.

Assessment - Exam [30%]

Summative

Final written exam that will assess specific parts of the outcomes that are not completely covered by the other assessment tools.

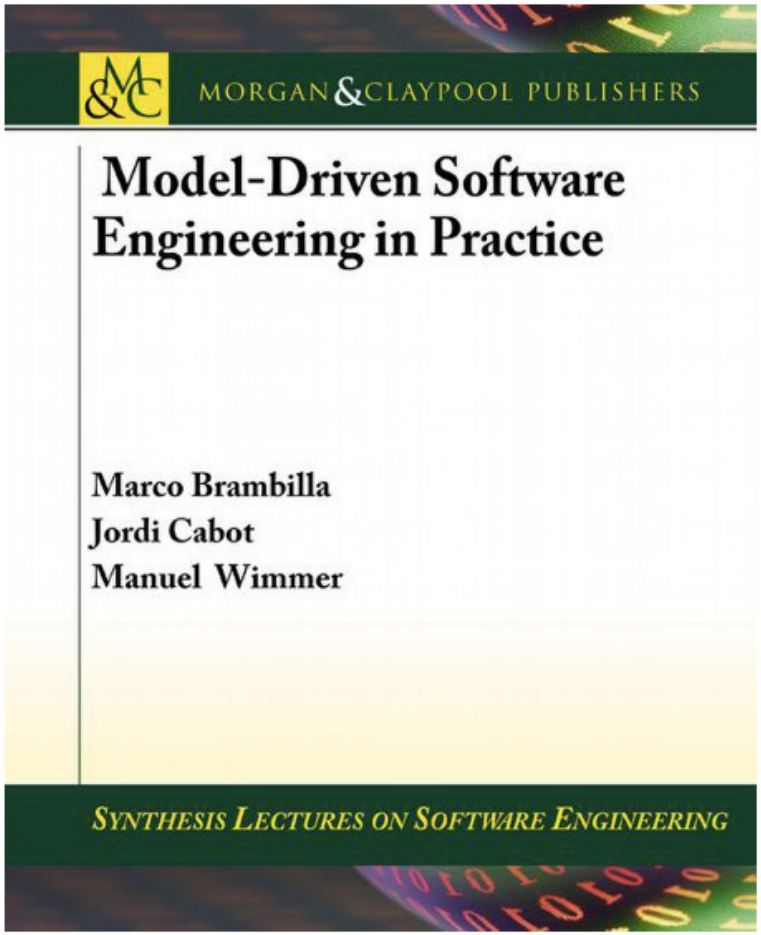
The exam will be composed of groups of questions of several types:

- True/False questions with justification;
- Short answer questions with justification;
- Essay Questions.

Notes:

- Questions of the exam will be according to the executed program of the course.
- Students will have knowledge about the weight of each question.
- The final score of the exam is given in the scale 0-20,0.
- The final exam has a minimum score of 8,0/20,0.

Main Book of the Course



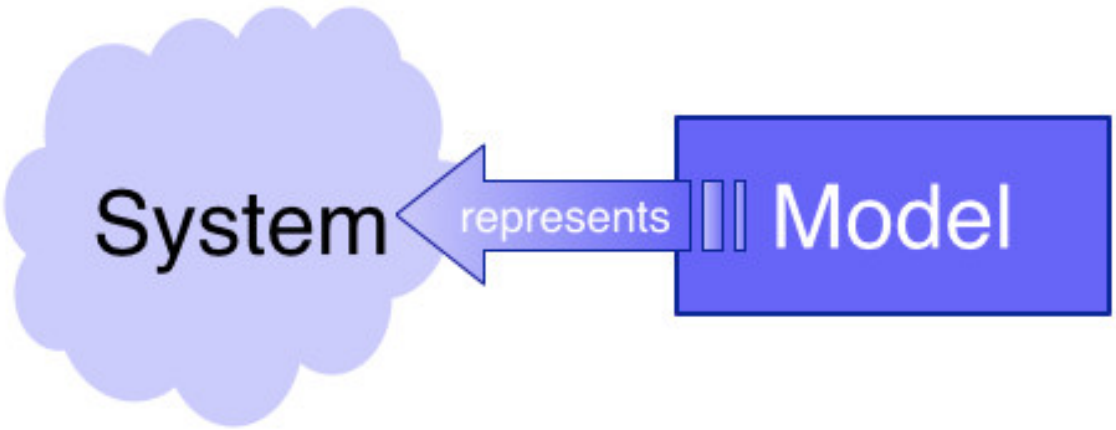
"Model-Driven Software Engineering in Practice, 2nd Edition", Marco Brambilla et al., Morgan & Claypool Publishers, 2017

- This is the main book for this course.
- The contents of this book will be used during several lectures.
- The book site: `http://mdse-book.com`

Motivation: Abstractions and Models

- The human mind continuously re-works reality by applying cognitive processes
- **Abstraction**: capability of finding the commonality in many different observations:
  - generalize specific features of real objects (generalization)
  - classify the objects into coherent clusters (classification)
  - aggregate objects into more complex ones (aggregation)
- **Model**: a simplified or partial representation of reality, defined in order to accomplish a task or to reach an agreement.

Models: What is a Model?



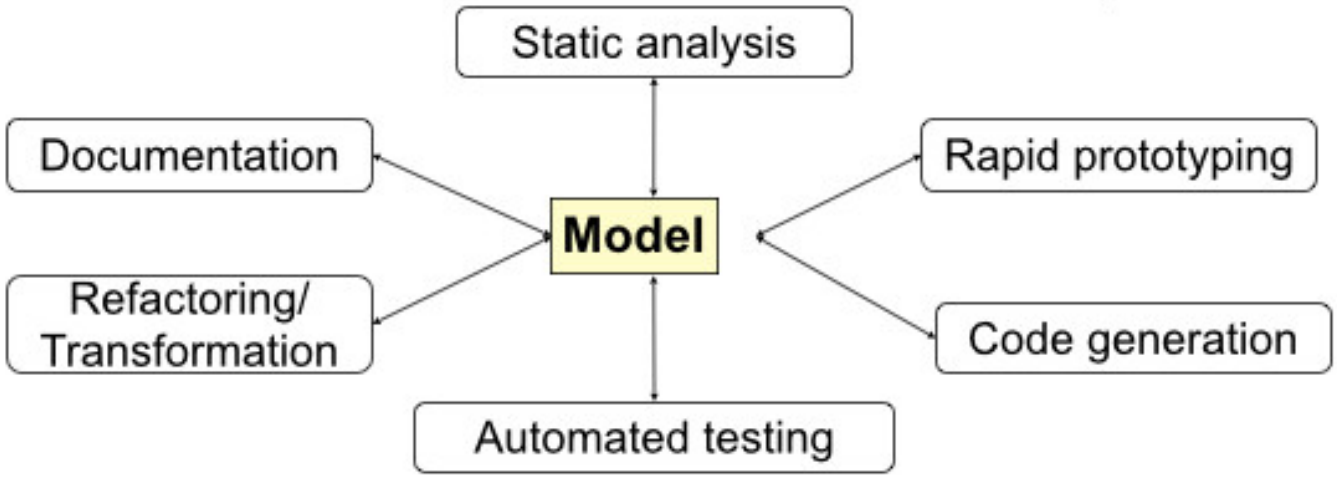
Mapping Feature	A model is based on an original (=system)
Reduction Feature	A model only reflects a (relevant) selection of the original's properties
Pragmatic Feature	A model needs to be usable in place of an original with respect to some purpose

Purposes:

- descriptive purposes
- prescriptive purposes

What is Model Engineering?

Model as the **central artifact** of software development



Related terms:

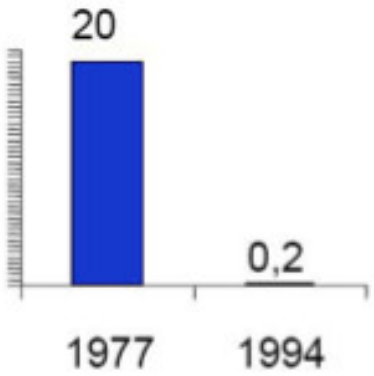
- Model Driven Engineering (MDE),
- Model Driven [Software] Development (MDD/MDSD),
- Model Driven Architecture (MDA)
- Model Integrated Computing (MIC)

Why Model Engineering?

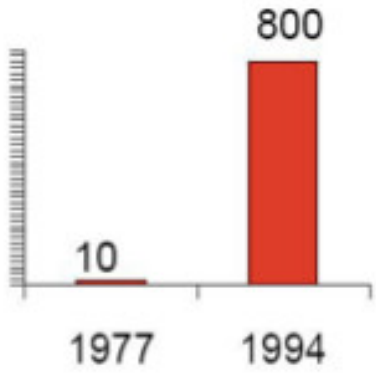
- Increasing complexity of software
  - Increasing basic requirements, e.g., adaptable GUIs, security, network capabilities, ...
  - Complex infrastructures, e.g., operating system APIs, language libraries, application frameworks
- Software for specific devices
  - Web browser, mobile phone, navigation system, video player, etc.
- Technological progress ...
  - Integration of different technologies and legacy systems, migration to new technologies
- ... leads to problems with software development
  - Software finished too late
  - Wrong functionality realized
  - Software is poorly documented/commented
  - and can not be further developed, e.g., when the technical environment changes, business model/ requirements change, etc.

Why Model Engineering?

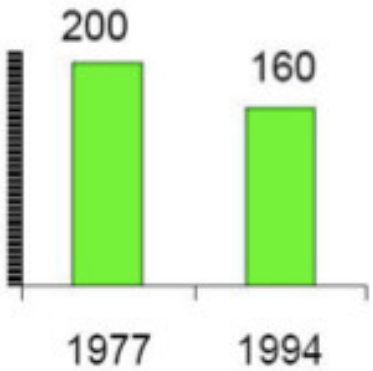
Quality problems in software development



Number of bugs per 1000 LOC



Program size (1000 LOC)



Resulting absolute bug count

Real quality improvements are only possible if the increase in program complexity is **overcompensated** !

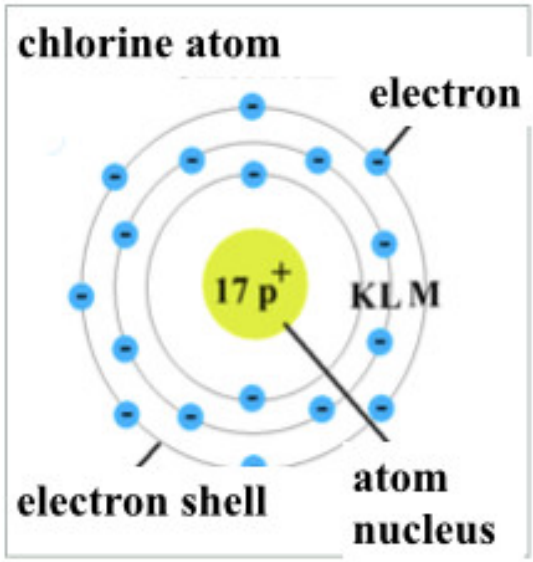
(Average values, from Balzert 96)

Why Model Engineering?

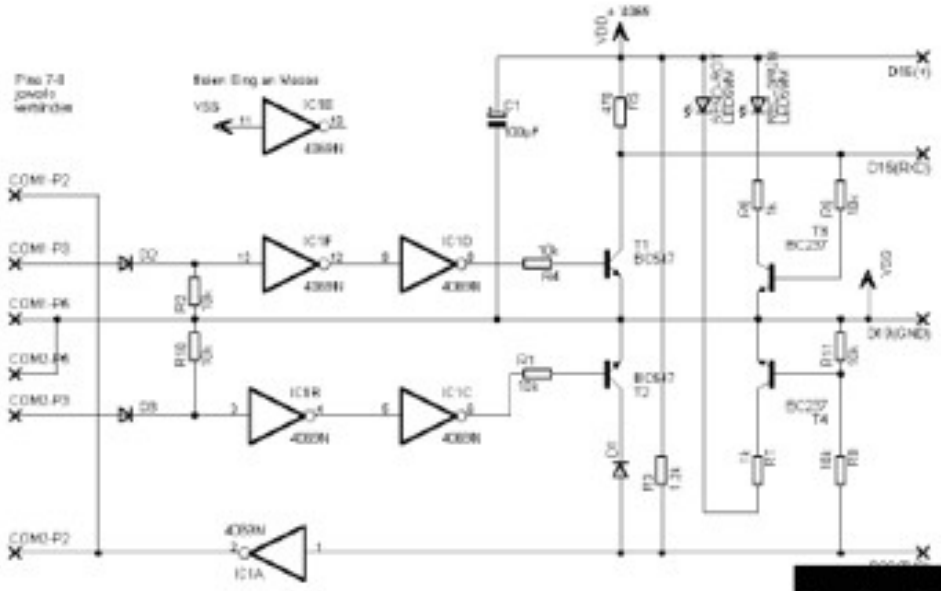
- **Traditional** usage of models in software development
  - **Communication** with customers and users (requirement specification, prototypes)
  - Support for software design, capturing of the **intention**
  - **Task specification** for programming
  - **Code visualization**, for example in TogetherJ
- What is the **difference** to Model Engineering?

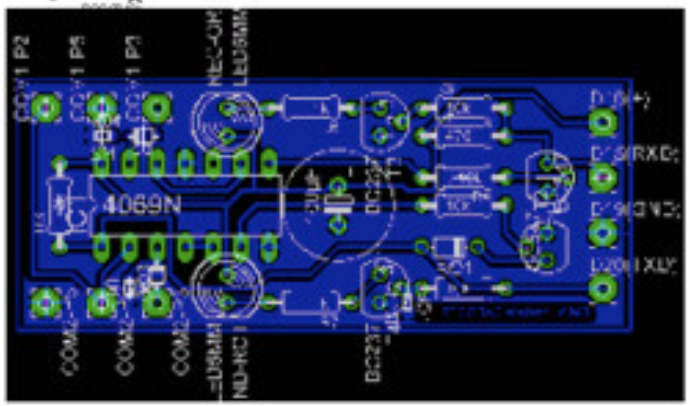
Usage of Models

- Do not apply models as long as you have not checked the underlying **simplifications** and evaluated its **practicability**.
- Never mistake the **model** for the **reality**.
  - Attention: abstraction, abbreviation, approximation, visualization, ...



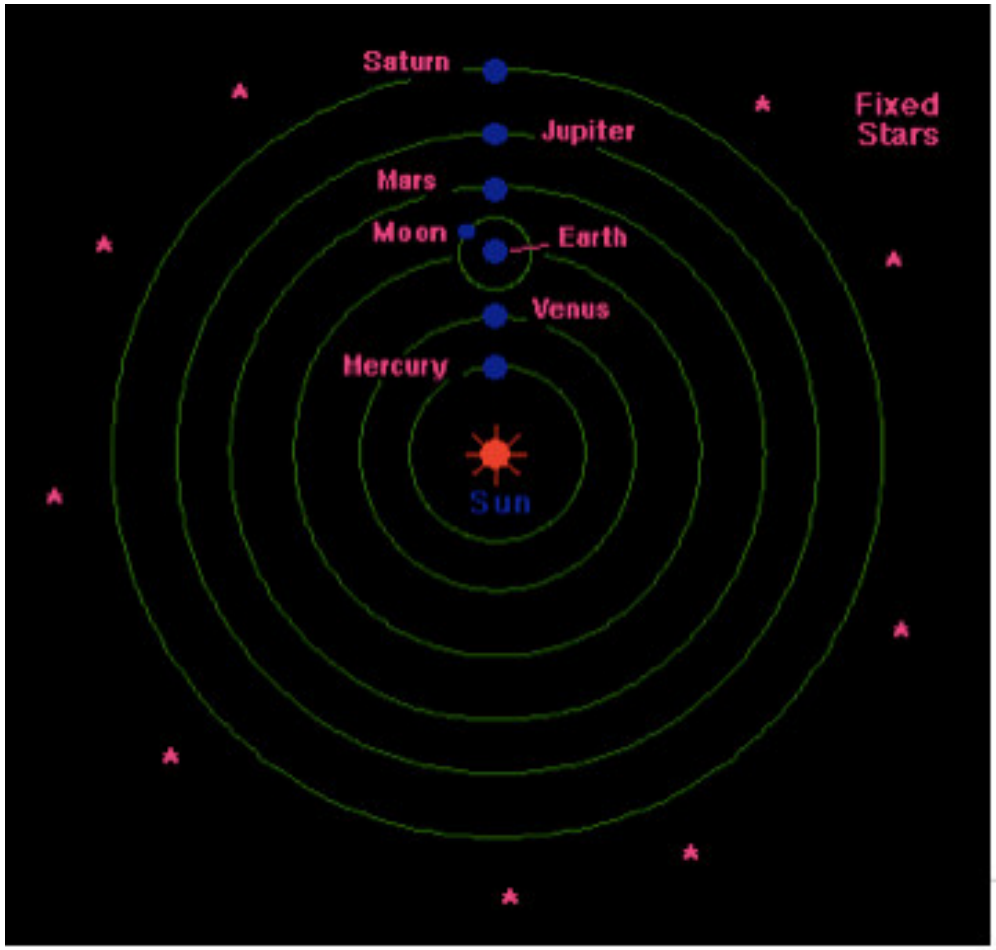
Constructive models (Example: Electrical Engineering)





Declarative models (Example: Astronomy)

Heliocentric model by Kopernikus

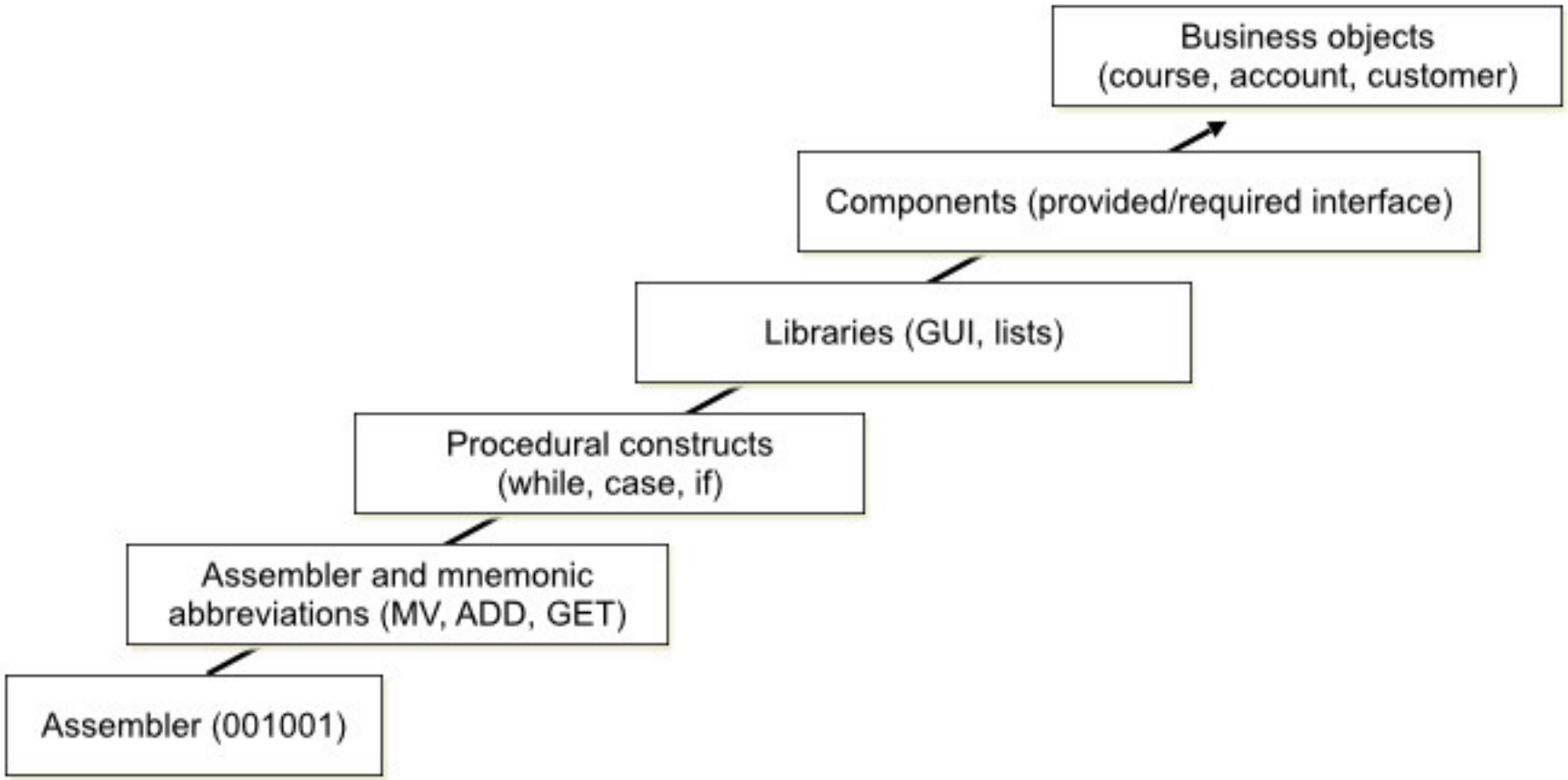


Application area of modeling

- **Models as drafts**
  - Communication of ideas and alternatives
  - Objective: modeling per se
- **Models as guidelines**
  - Design decisions are documented
  - Objective: instructions for implementation
- **Models as programs**
  - Applications are generated automatically
  - Objective: models are source code and vice versa

Increasing abstraction in software development

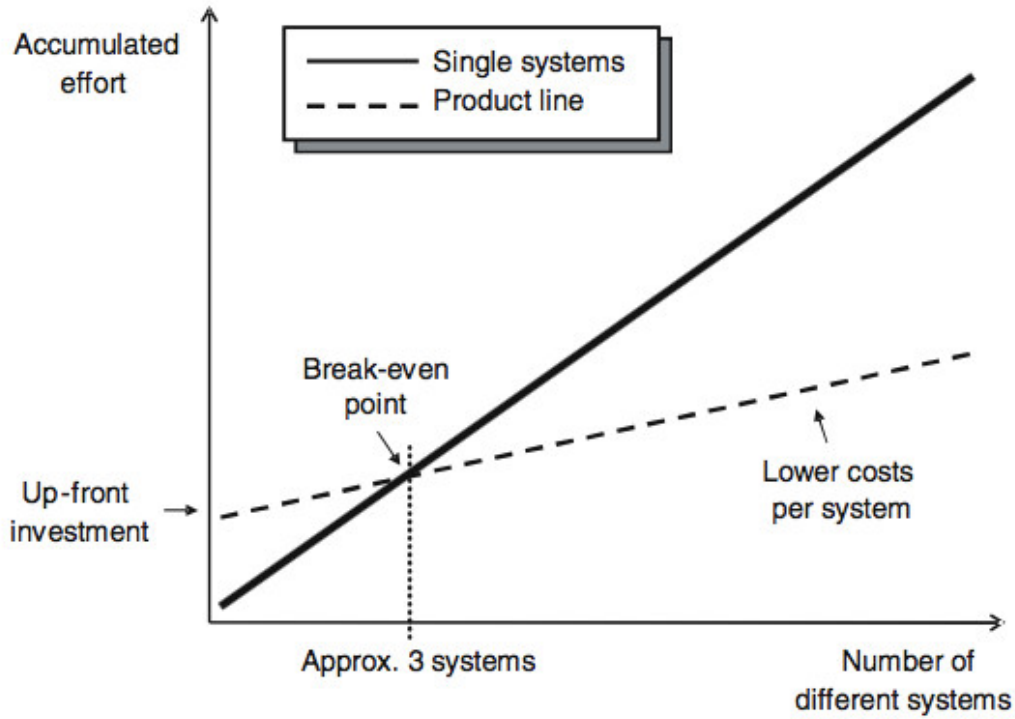
The **used artifacts of software development** slowly converge to the concepts of the **application area**.
In another perspective, we have also being increasing how we are able to **reuse constructs in our software solutions**.



Industrial Reuse -> Software Product Lines

A software product line is a set of software-intensive systems ***sharing a common, managed set of features*** *that satisfy the specific needs of a* ***particular market segment or mission*** *and that are developed from a* ***common set of core assets in a prescribed way***[2].

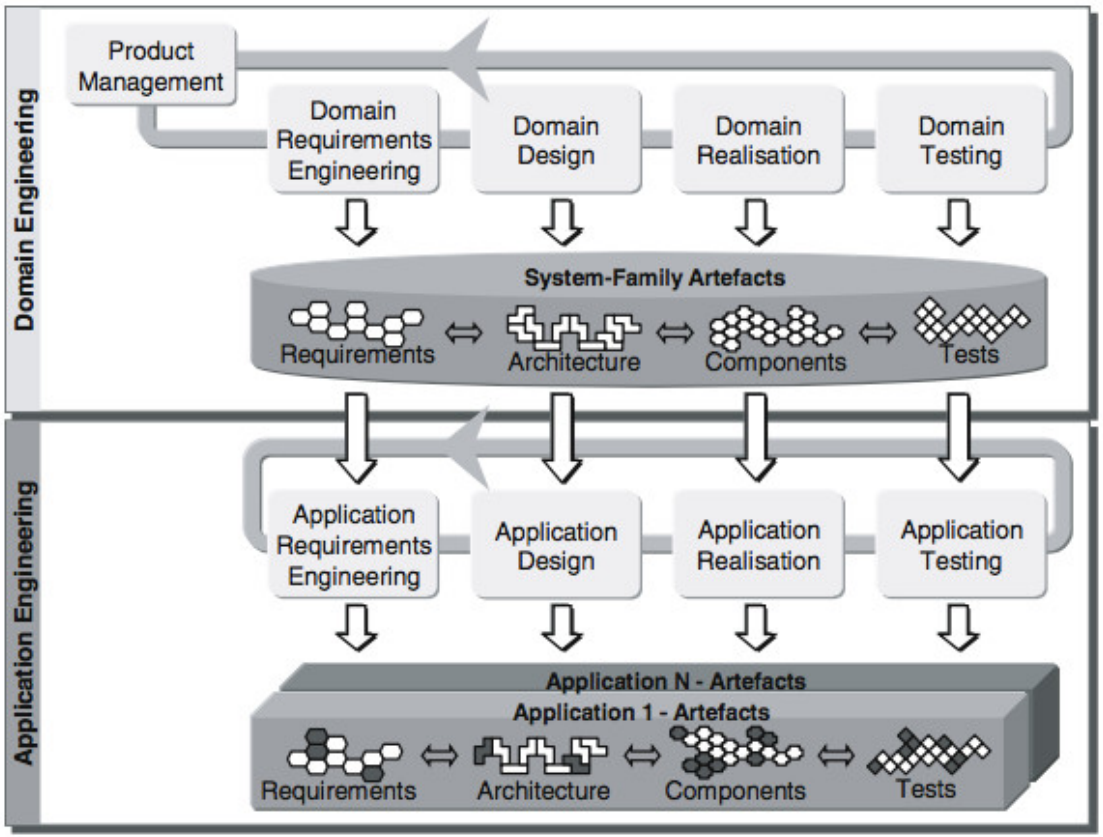
Figure: Economics of software product line engineering[3]



[2] From "Software Product Lines: Practices and Patterns", Paul Clements, Linda Northrop, Addison-Wesley, 2001

[3] From "Software Product Lines in Action", Frank van der Linden et al, Springer, 2007

Figure: The two-life-cycle model of software product line engineering[4]



[4] From "Software Product Lines in Action", Frank van der Linden et al, Springer, 2007

Software Product Lines Fundamental Principles

- **Variability management**: individual systems are considered as variations of a common theme. This variability is made explicit and must be systematically managed.
- **Business-centric**: software product line engineering aims at thoroughly connecting the engineering of the product line with the long-term strategy of the business.
- **Architecture-centric**: the technical side of the software must be developed in a way that allows taking advantage of similarities among the individual systems.
- **Two-life-cycle approach**: the individual systems are developed based on a software platform. These products – as well as the platform – must be engineered and have their individual life-cycles.