Summer Term 2018

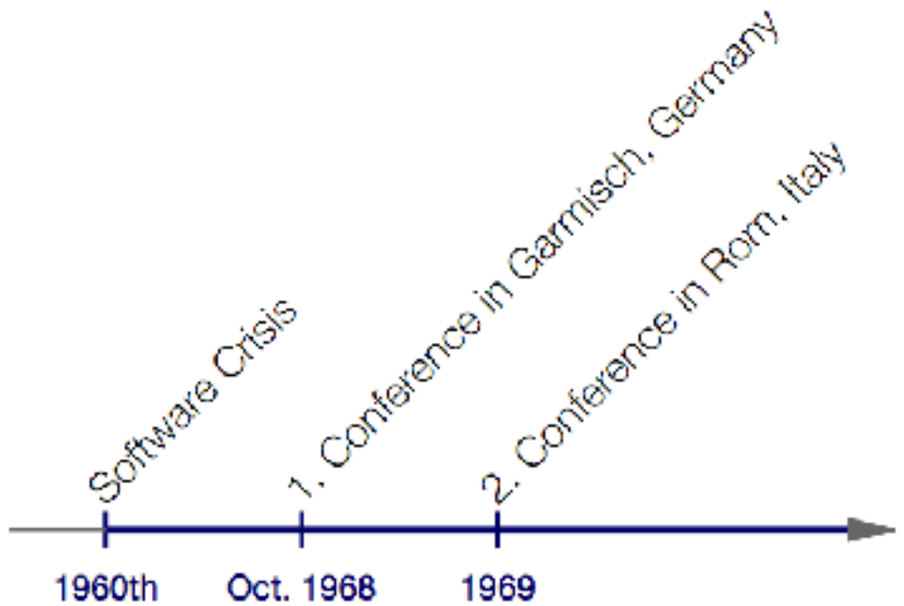
Software Engineering Design & Construction

Dr. Michael Eichberg
Fachgebiet Softwaretechnik
Technische Universität Darmstadt

Introduction - Software Engineering

Software Engineering Timeline

Impressions from the NATO Software Engineering Conferences



[The major cause of the software crisis is] that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.

E. Dijkstra, 1972

Definition of Software Engineering

- Carnegie Mellon University's Software Engineering Institute (SEI) defines "*Software Engineering*" relative to "*Engineering*" as:
  - *Engineering* is the systematic application of scientific knowledge in creating and building cost-effective solutions to practical problems in the service of mankind.
- *Software engineering* is that form of engineering that *applies the principles of computer science and mathematics* to achieving cost-effective solutions to software problems.

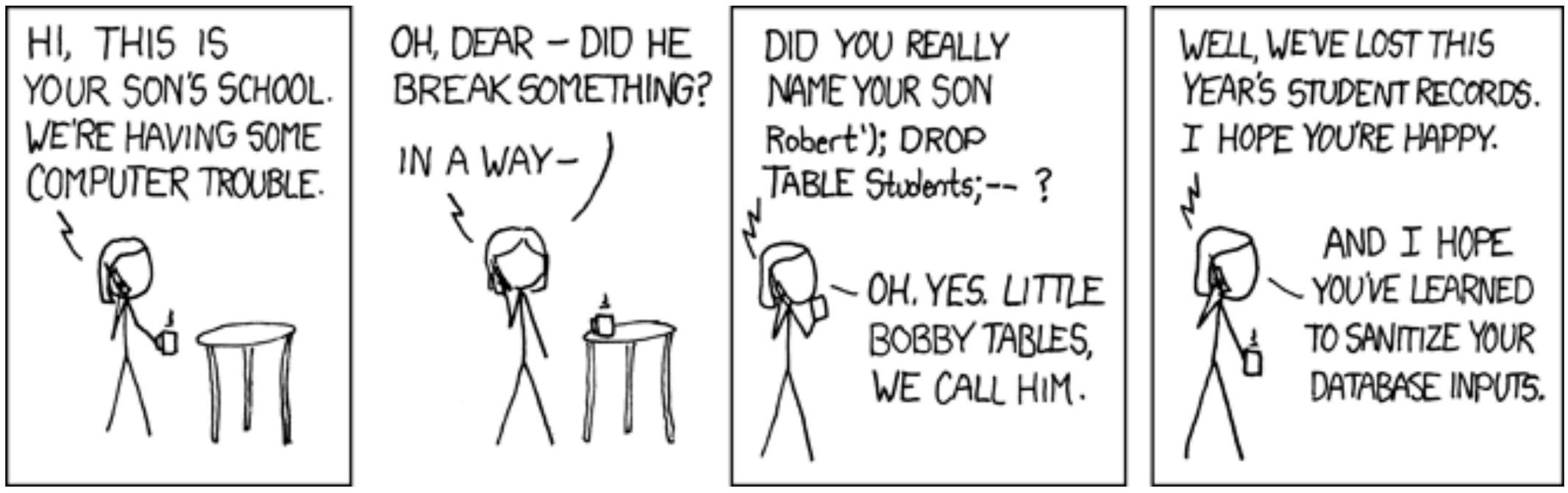
But, as Dijkstra already identified ....
[...] I would like to insert a warning to those who identify the difficulty of the programming task with the struggle against the inadequacies of our current tools, because they might conclude that, once our tools will be much more adequate, programming will no longer be a problem.

Programming will remain very difficult, because once we have freed ourselves from the circumstantial cumbersomeness, we will find our selves free to tackle the problems that are now well beyond our programming capacity.

E. Dijkstra, 1972

New Programming Concepts to solve complex tasks:
e.g., Reactive Programming

“…struggle against the inadequacies of our current tools…”



http://imgs.xkcd.com/comics/exploits_of_a_mom.png

2477

Vulnerabilities due to buffer errors (2013-2015)

National Vulnerability Database, hKp://nvd.nist.gov

On the State of Software Engineering

2230

Vulnerabilities due to cross-site scripting (2013-2015)

National Vulnerability Database, hKp://nvd.nist.gov

On the State of Software Engineering

2230

Vulnerabilities due to permissions, privileges and access control (2013-2015)

National Vulnerability Database, hKp://nvd.nist.gov

On the State of Software Engineering

1769

Vulnerabilities due to cryptographic issues (2013-2015)

National Vulnerability Database, hKp://nvd.nist.gov

On the State of Software Engineering

[…] 87.8% of the applications present some kind of [crypto API] misuse […]

A. Chatzikonstantinou, C. Ntantogian, G. Karopoulos, and C. Xenakis. "*Evaluation of Cryptography Usage in Android Applications*". In: International Conference on Bio-inspired Information and Communications Technologies. 2016

These issues were first described in…

- 1972 - Buffer overflow used in a kernel
  Computer Security Technology Planning Study, 1972

- 1988 - Buffer overflow used in the Morris worm

- 1998 - SQL injection explained in the literature
  Phrack Magazine, 8(54), article 8

- 2000 - Cross-site scripting exploits
  CERT “Malicious HTML Tags”, 2000

Product Engineering

Software as an Engineering Product?

- **1st Phase**: *Requirements Analysis*
  The problem to solve is analyzed and documented.

- **2nd Phase**: *Design and Validation*
  Engineers translate the requirements into a detailed description of the solution using models and rigorously validate these models.

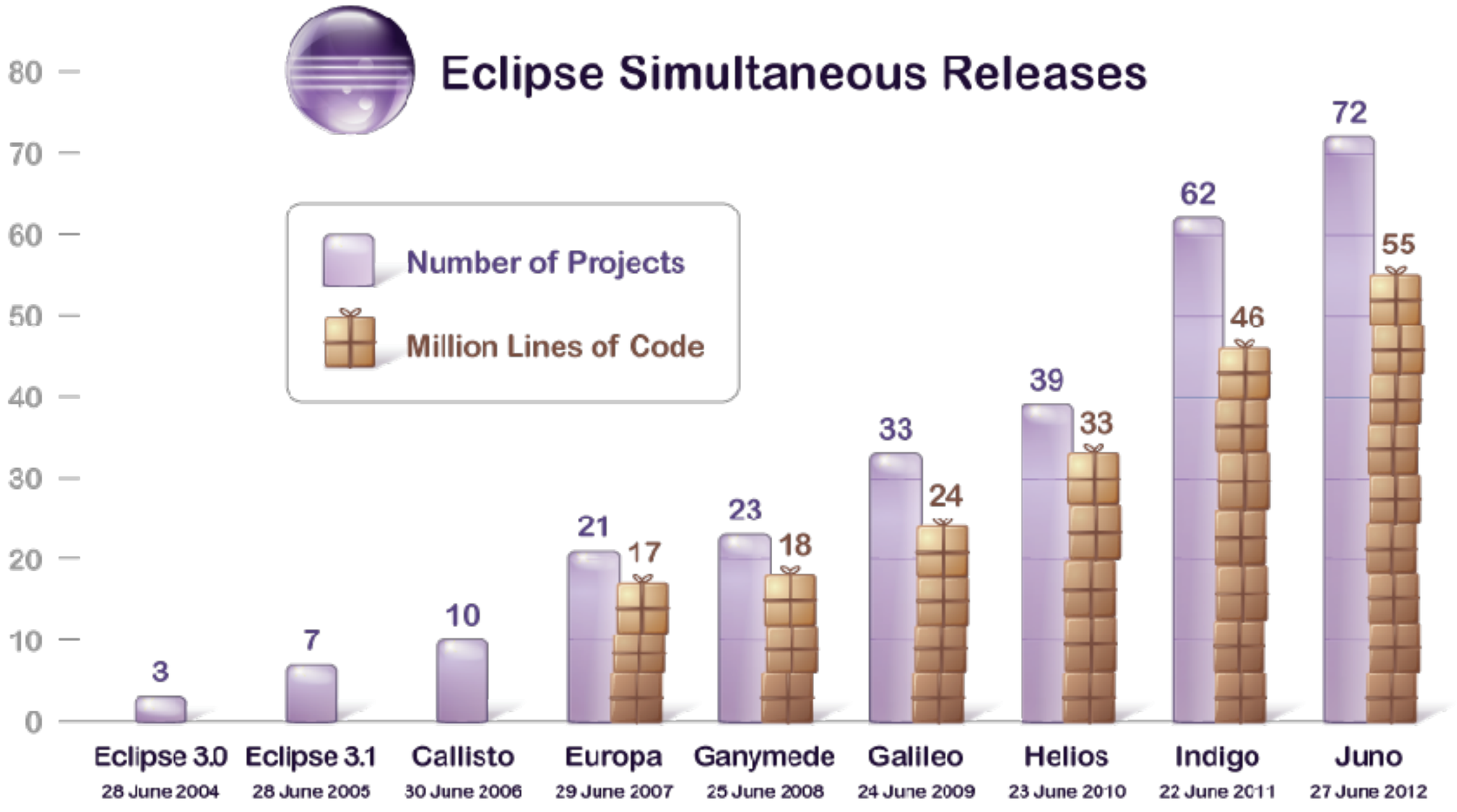
- **3rd Phase**: *Building the Product*
  Workers build the design using appropriate tools and materials.

Software as an Engineering Product?

Hardware vs. Software Design

Hardware Design	Software Design
Product is a physical object	Product is the running software
Building the product is: • done by humans and robots • expensive • slow • hard to redo	Building the product is: • done by compilers and linkers • extremely cheap • very fast • easy to redo
Precise quality measures	No precise quality measures

Development of Eclipse



- *Code as Design*: Three Essays by Jack W. Reeves
  http://www.developerdotstar.com/mag/articles/reeves_design_main.html
  The following essays offer three perspectives on a single theme, namely that **programming is fundamentally a design activity** and that the only final and true representation of "the design" is the source code itself. [...]

- *What Is Software Design?*
  http://www.developerdotstar.com/mag/articles/reeves_design.html
  This essay was first published in the Fall 1992 issue of the C++ Journal. [...] in recent years the essay has entered the flow of ideas and discussion in the software development community at large [...].

- *What Is Software Design: 13 Years Later*
  http://www.developerdotstar.com/mag/articles/reeves_13yearslater.html
  In this essay the author responds to the most common arguments he has encountered. He also considers certain ideas from the original essay in light of more current trends and techniques.

- *Letter to the Editor*
  http://www.developerdotstar.com/mag/articles/reeves_originalletter.html
  This is the original letter written by Jack W. Reeves to C++ Journal. It stands as a rewarding essay in its own right, giving first written expression to the themes and ideas found in "What is Software Design?" In some aspects it is even more comprehensive and spirited than the essay it inspired.

NATO Software Engineering Conference 1968

- *[...] there is no essential difference between design and production, since the production will include decisions which will influence the performance of the software system, and thus properly belong in the design phase.*

  Peter Naur

- *[…] Honestly, I cannot see how these activities allow a rigid separation if we are going to do a decent job.*

  Edsger Dijkstra

Back in the 1960s, writing the source code was considered to be the "production step".

Questionable Ideas

Grounded on Misleading Analogies

Software Industrialization

Waterfall Model

Programmer productivity as "Lines of Code"

Outsourcing: Design here, Production elsewhere

Consequences of the Cheap Software Build?

In Software Development Complexity and Change are Invariants!

In Software Development **Complexity and Change are Invariants!**

Designing for *organizing complexity*
and *facilitating change*
is the key to support maintainability.

Incomplete and Changing Requirements

What is the right question to ask?

- How do we get complete requirements?
- Can we get complete requirements at all?

Incomplete and Changing Requirements

- Scenario:
  You are developing a software for personnel management that advices employees about their benefits, including their retirement plan.

- Initial Requirement:
  The first opportunity to withdraw money without penalty is when an employee turns 60.

- Resulting Code:
  `if(employee.age >= 60) {...}`

Does this reflect the reality?

Incomplete and Changing Requirements

- Scenario: (as previously shown)
- Initial Requirement: (as previously shown)
- Resulting Code:

```
if(employee.age >= 60) {…}
```

- Changed Requirement:
  After testing it is discovered that withdrawing money is possible when the employee is 59.5.
- Customer's Assessment: This change will be easy to do...
- But, the following change may not be possible:

```
if(employee.age >= 59.5) {...}
```

Does this reflect the reality?

Incomplete and Changing Requirements

- Scenario: (as previously shown)
- In
- R
- Changed Requirement: After testing it is discovered that withdrawing money is p
employee is 59.5.
- Customer's Assessment: This change will be easy to do...
- But, the following change may not be possible:

```
if(employee.age >= 59.5) {...}
```

The task of the software development team is to engineer the illusion of simplicity.

in G. Booch: Object-oriented Analysis with Applications, Addison-Wesley, 1993

Does this reflect the reality?

Everything is Part of the Design

1. High-level architectural design
2. Class-level design
3. Low(implementation)-level detailed design
4. **The code**

Testing and Debugging is Designing

Testing is not just concerned with getting the current design correct, it is part of refining the design.

In […] embedded systems […] the software requirements change throughout the software development process, even during system testing. This is largely due to unanticipated behavior, dynamic changes in the operating environment, and complex software/hardware and software/software interactions in the systems being developed.

Controlling requirement changes (and, hence, the scope and cost of development) is difficult since the changes are often prompted by an ***improved understanding of the software's necessary interfaces*** *with the physical components […] in which it is embedded.*

Analyzing software requirements errors in safety-critical, embedded systems; R. R. Lutz; 1993

Auxiliary Documentation

- Auxiliary documentation should capture information from the problem space.

- Auxiliary documentation should document those aspects of the design that are difficult to extract directly from the design itself.

- But, keeping auxiliary documentation up to date manually is difficult.

"Up-to-Date" Auxiliary Documentation

```
package org.eclipse.emf.teneo.hibernate.resource;
...
  /**
   * Rolls back the transaction if any and clears different lists to
   * start with an empty resource again.
   * Note that the super.doUnload is not called because that clears
   * the list resulting in all kinds of undesirable inverse removes.
   */
  @Override
  protected void doUnload() {
     super.doUnload();
  }
...
```

TO CODE IS TO DESIGN.

Which doesn't mean that you should start coding right away!

Properties of a good development process

- … recognizes *programming as a design activity* and does not hesitate to code when coding makes sense.

- … recognizes *testing and debugging as design activities* - the software equivalent of design validation and refinement of other engineering disciplines - and does not shorten the steps.

- … recognizes that *there are other design activities*: top level design, module design, class design, etc.

- … recognizes that *all design activities interact* and allows the design to change as various steps reveal the need.

If *to code is to design*, then

Programming Languages

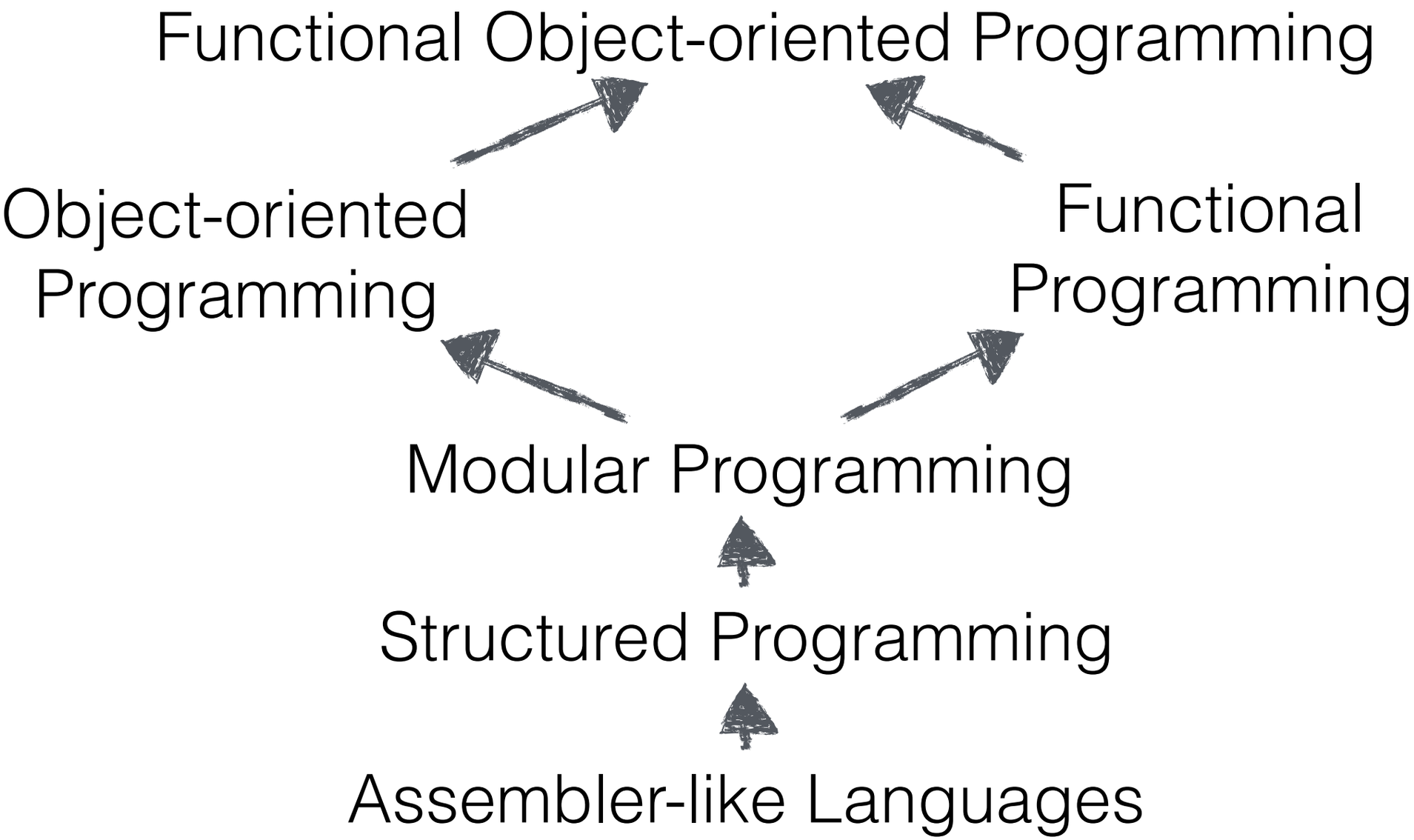
are

Design Languages!

Which doesn't mean that you should start coding right away!

We need a *unified design notation suitable for all levels of design*, i.e., programming languages that are suitable for capturing high-level design.

Making Code Look Like Design



Advances in programming language technology are driven by the need to **make** programming languages capable of more directly capturing higher-level designs!

Takeaway

Consequences of the cheap build for this course

- Designing means structuring code in modular way so as to support managing complexity and continuous change.

- We will adopt an agile design process to accommodate change in the design process.

- We will adopt test-driven development, as we consider testing to be part of design.

- Languages as design notations will be in focus but also design principles and styles as well as tools for expressing modular structures outside the languages.