b
UNIVERSITÄT
BERN

Introduction to Software Engineering (ESE : Einführung in SE)

Mircea F. Lungu

Based on a lecture by Oscar Nierstrasz.
Selected material courtesy of Serge Demeyer.

ESE — Introduction

Lecturer	Mircea F. Lungu scg.unibe.ch/staff/mircea
Assistants	Andrea Caracciolo Andrei Chiș, Bledar Aga
Lectures	Engehaldenstrasse 8, 001, Wednesdays @ 13h15-15h00
Exercises	Engehaldenstrasse 8, 001 Wednesdays @ 15h00-16h00
WWW	scg.unibe.ch/teaching/ese

Roadmap

- Course Overview
- What is Software Engineering?
- Development Lifecycles
- Software Development Activities
- Methods and Methodologies

Roadmap

- Course Overview
- What is Software Engineering?
- Development Lifecycles
- Software Development Activities
- Methods and Methodologies

Principle Texts

Software Engineering. Ian Sommerville. Addison-Wesley Pub Co; ISBN: 020139815X, 7th edition, 2004

Software Engineering: A Practioner's Approach. Roger S. Pressman. McGraw Hill Text; ISBN: 0072496681; 5th edition, 2001

Using UML: Software Engineering with Objects and Components. Perdita Stevens and Rob J. Pooley. Addison-Wesley Pub Co; ISBN: 0201648601; 1st edition, 1999

Designing Object-Oriented Software. Rebecca Wirfs-Brock and Brian Wilkerson and Lauren Wiener. Prentice Hall PTR; ISBN: 0136298257; 1990

Recommended Literature

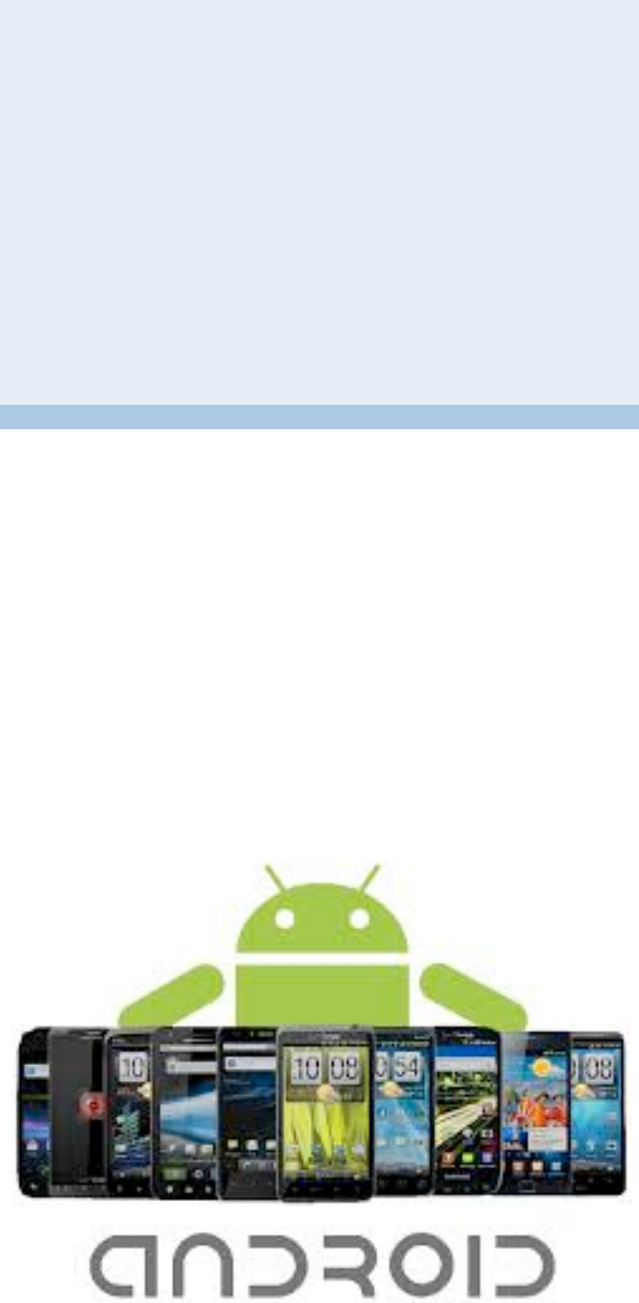
- *eXtreme Programming Explained: Embrace Change.* Kent Beck. Addison-Wesley Pub Co; ISBN: 0201616416; 1st edition (October 5, 1999)
- *The CRC Card Book*. David Bellin and Susan Suchman Simone. Addison-Wesley Pub Co; ISBN: 0201895358; 1st edition (June 4, 1997)
- ***The Mythical Man-Month: Essays on Software Engineering***. Frederick P. Brooks. Addison-Wesley Pub Co; ISBN: 0201835959; 2nd edition (August 2, 1995)
- *Agile Software Development*. Alistair Cockburn. Addison-Wesley Pub Co; ISBN: 0201699699; 1st edition (December 15, 2001)
- ***Peopleware: Productive Projects and Teams***. Tom Demarco and Timothy R. Lister. Dorset House; ISBN: 0932633439; 2nd edition (February 1, 1999)
- *Succeeding with Objects: Decision Frameworks for Project Management.* Adele Goldberg and Kenneth S. Rubin. Addison-Wesley Pub Co; ISBN: 0201628783; 1st edition (May 1995)
- *A Discipline for Software Engineering.* Watts S. Humphrey. Addison-Wesley Pub Co; ISBN: 0201546108; 1st edition (December 31, 1994)

Lecture schedule

18 Sep	Introduction
25 Sep	Requirements Collection
2 Oct	Guest Lecture: Processes & Scrum
9 Oct	Usability and UI
16 Oct	Architecture
23 Oct	Good Software Design
30 Oct	Validation and Testing
6 Nov	Documenting Design with UML
13 Nov	Software Quality
20 Nov	Software Assessment Tutorial
27 Nov	Software Evolution
4 Dec	Empirical Software Engineering
11 Dec	*TBD*
18 Dec	Project showcase
7 Jan	Final Exam

Project

- **Goal**: Android application
- **Customers**: ESE Staff
- **Organization:** teams
- **Target**: The Android Market
- **Deadline**: December 18
- **Worth**: 40% of final grade



Grading

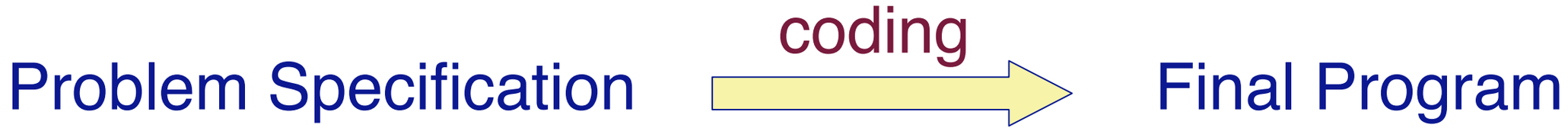
Final Grade = 0.4 * Project + 0.6 * Exam

Project =
0.1 * Cross-Project Code Review +
0.4 * Code/Design Quality & Documentation +
0.4 * Functionality & Usability (lack of bugs, richness of features) +
0.1 * Final Presentation

Roadmap

- Course Overview
- **What is Software Engineering?**
- Development Lifecycles
- Software Development Activities
- Methods and Methodologies

Why Software Engineering?



But ...

- Where did the specification come from?
- How do you know the specification corresponds to the user's needs?
- How did you decide how to structure your program?
- How do you know the program actually meets the specification?
- How do you know your program will always work correctly?
- What do you do if the users' needs change?
- How do you divide tasks up if you have more than a one-person team?

What is Software Engineering?

Software engineering is <u>different</u> from other engineering disciplines.

— Sommerville

Not constrained by physical laws
limit = human mind

It is constrained by political forces
balancing stake-holders

What is Software Engineering? (Def: I)

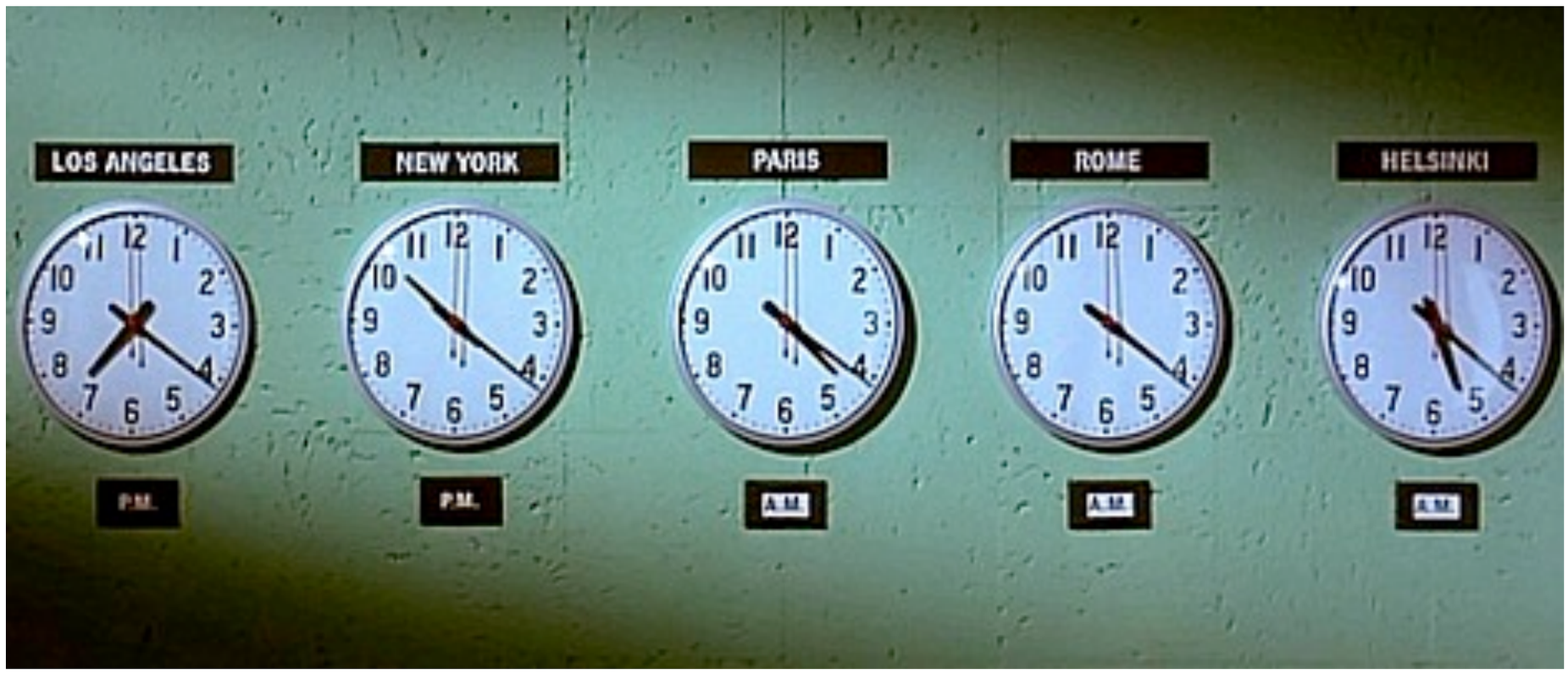
State of the art of developing quality software on time and within budget

(Anonymous)

Trade-off between perfection and physical constraints
- SE has to deal with real-world issues

State of the art!
- Community decides on “best practice” + life-long education

What is Software Engineering? (Def: II)



Multi-person construction of multi-version software.

— Parnas

Team-work

Scale issue (“program well” is not enough) + Communication Issue

Successful software systems must evolve or perish

Change is the norm, not the exception

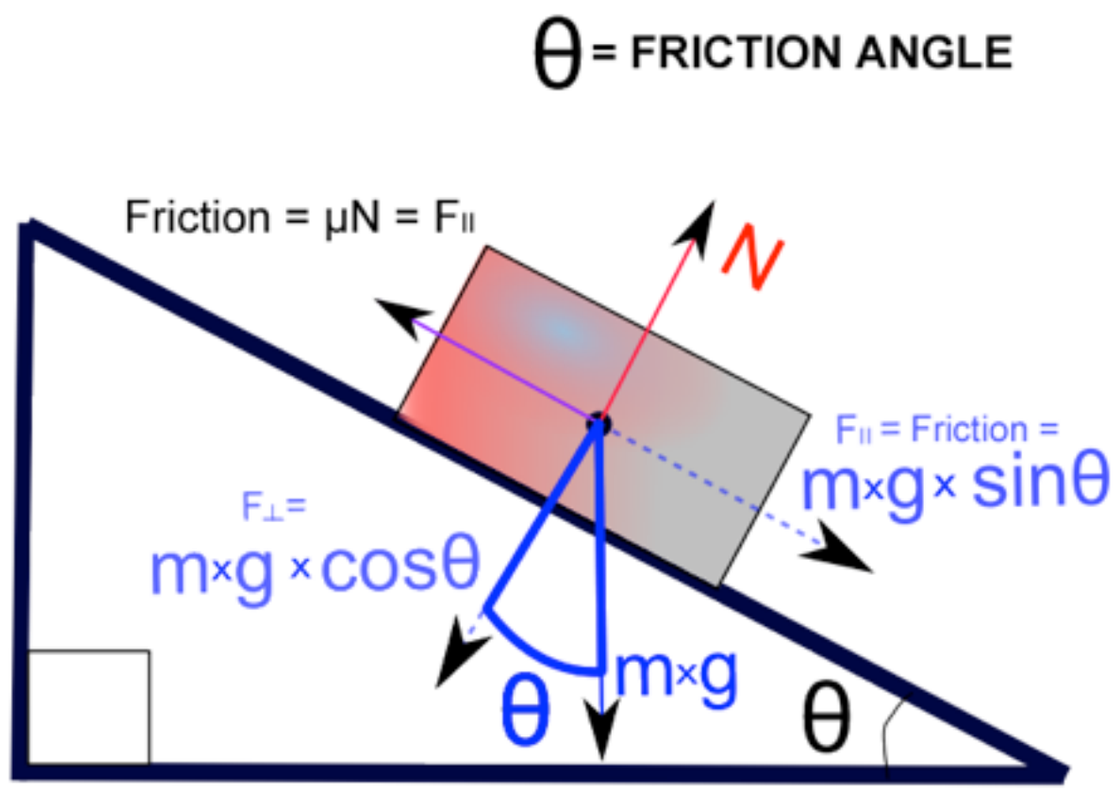
What is Software Engineering? (Def: III)

Everything else, besides programming, that contributes to building software systems. *Plus programming*.

This definition is mine, not of a famous person. But it is important to accentuate the fact that a successful system is a function of much more than programming.

Every one of you should be able to implement Facebook by the end of your bachelor. The reason why there is only one Facebook is everything else they did right besides programming. Plus luck.

Software Engineering and Computer Science



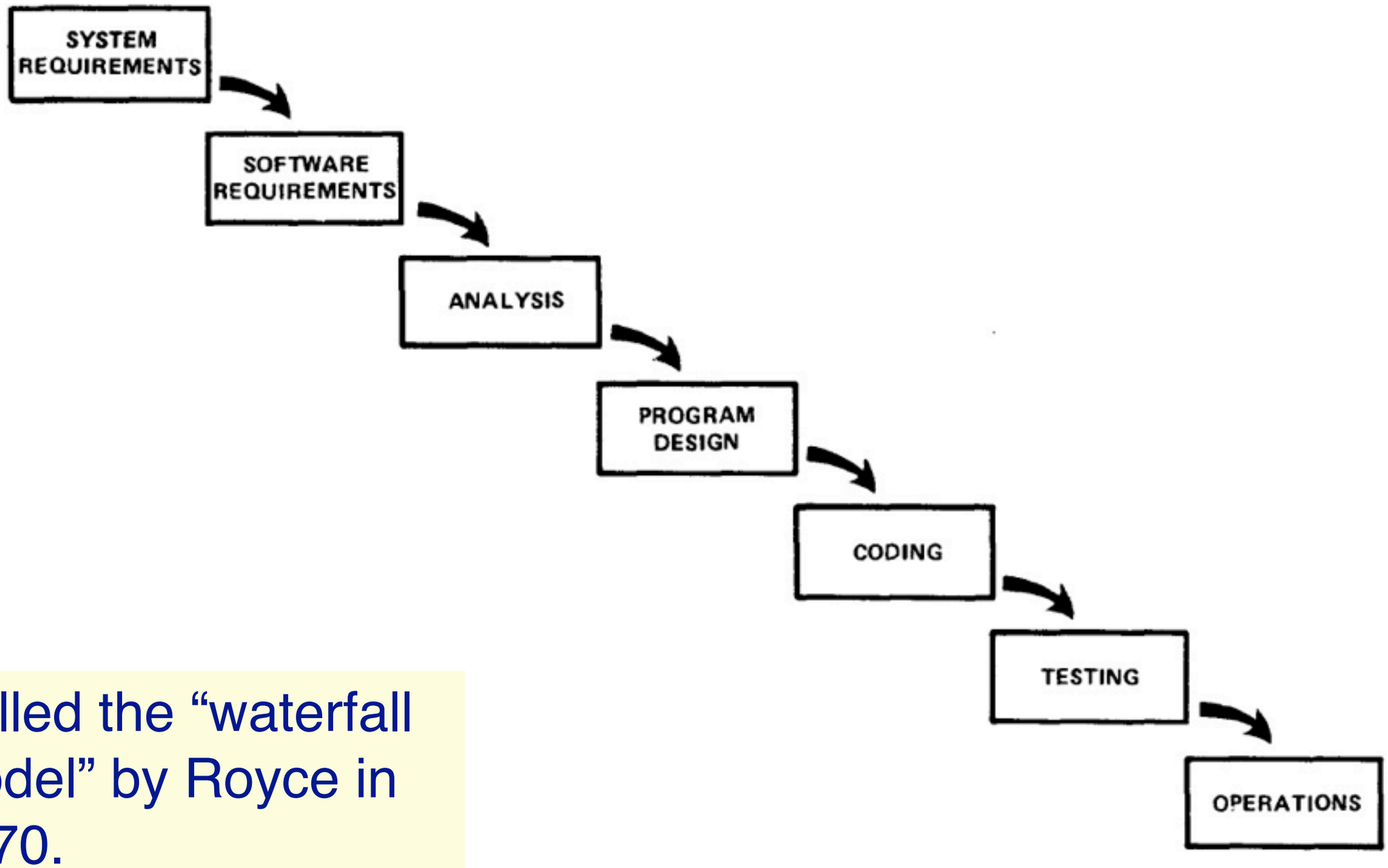
The relationship between software engineering and computer science is the same as that between mechanical engineering disciplines and physics. One supports the other. Only that the difference between CS and SE is much smaller.

http://commons.wikimedia.org/wiki/File:Mechanical_engineer_G._C._van_Vlaanderen_making_some_adjustments_to_previous_model_of_cable_cars_in_1973.jpg

Roadmap

- Course Overview
- What is Software Engineering?
- **Development Lifecycles**
- Software Development Activities
- Methods and Methodologies

The Classical Software Lifecycle



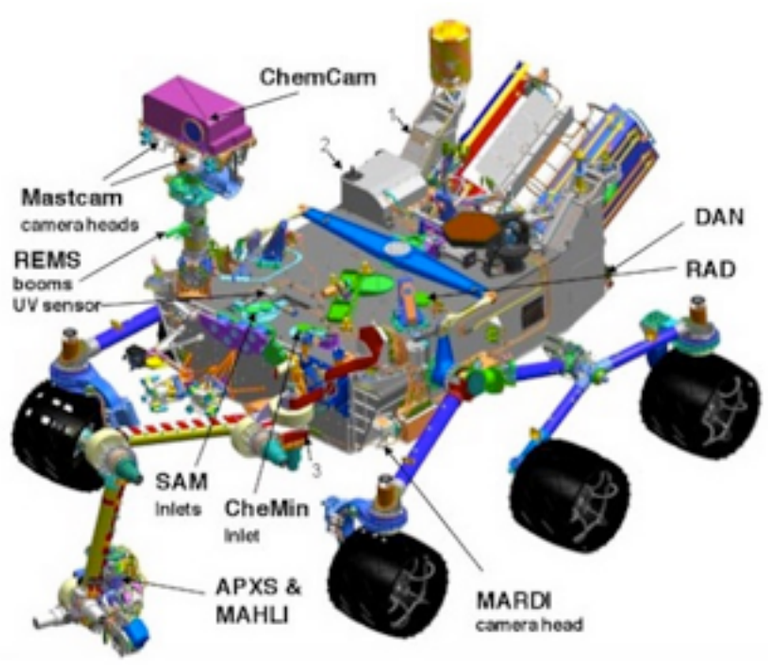
Called the “waterfall model” by Royce in 1970.

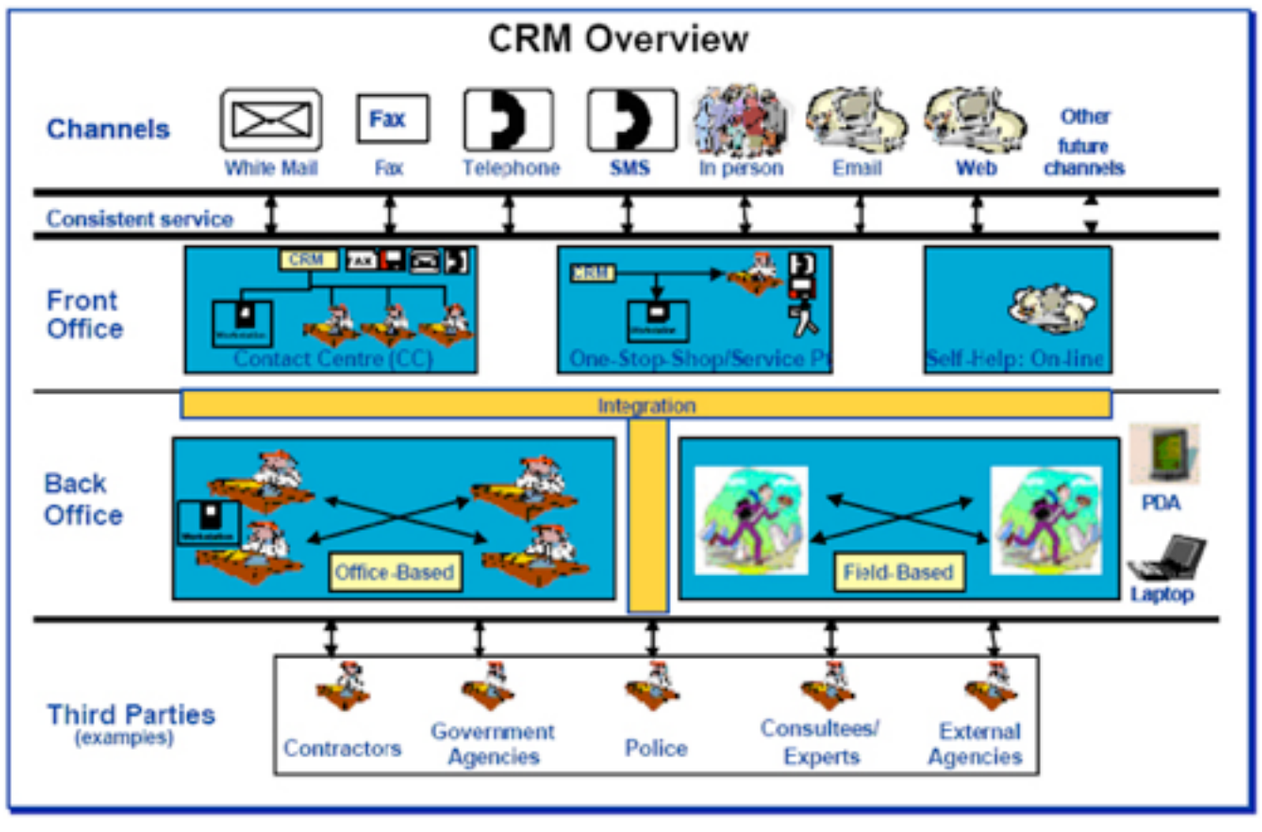
Software Development Activities

Requirements Collection	Establish customer’s needs
Analysis	Model and specify the requirements (“what”)
Design	Model and specify a solution (“how”)
Implementation	Construct a solution in software
Testing	Validate the solution against the requirements
Maintenance	Repair defects and adapt the solution to new requirements

The classical software lifecycle models the software development as a step-by-step “waterfall” between the various development phases.

Problems with the Waterfall Lifecycle





The waterfall model is often unrealistic for many reasons:

“Real projects rarely follow the sequential flow that the model proposes. Iteration always occurs and creates problems in the application of the paradigm”

“It is often difficult for the customer to state all requirements explicitly. The classic life cycle requires this and has difficulty accommodating the natural uncertainty that exists at the beginning of many projects.”

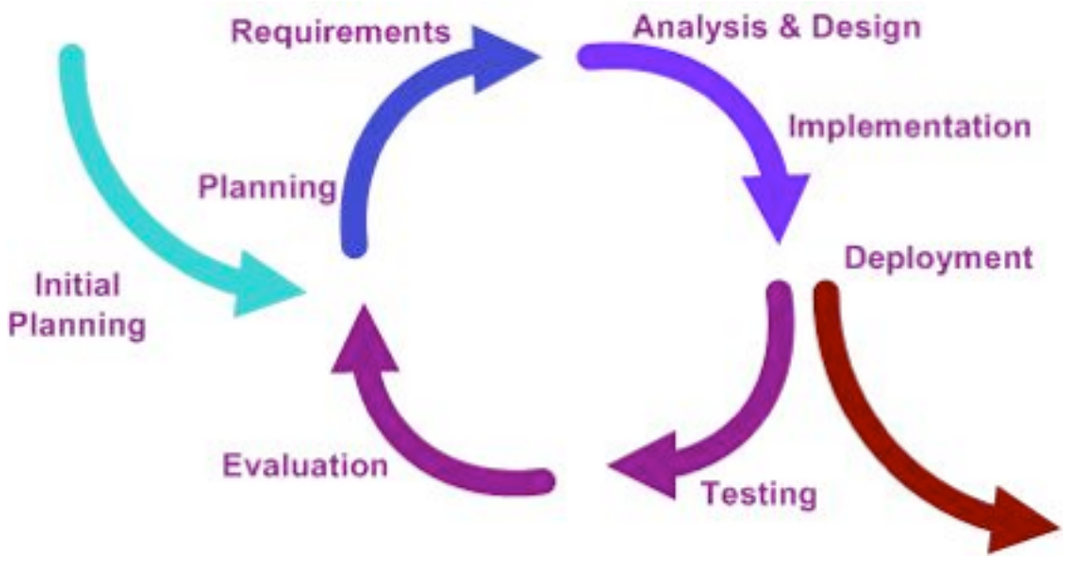
“The customer must have patience. A working version of the program(s) will not be available until late in the project timespan. A major blunder, if undetected until the working program is reviewed, can be disastrous.”
— Pressman, SE, p. 26

Requirements must be frozen too early in the life-cycle and the implementation is validated too late

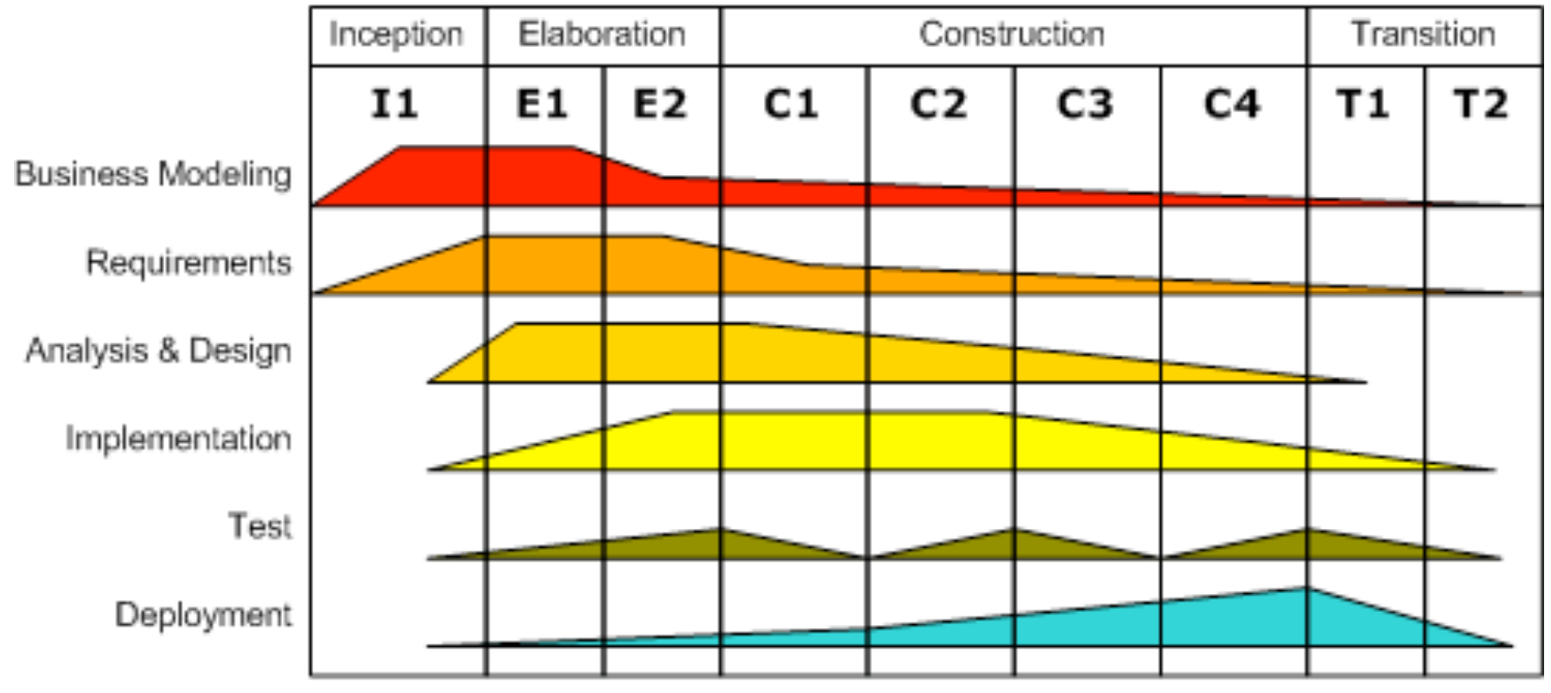
However, this depends on the type of the application. When one is implementing the Mars rover, things might be different.

Iterative Development

How do you plan the number of iterations?
How do you decide on completion?



Iterative Development
Business value is delivered incrementally in time-boxed cross-discipline iterations.



In practice, development is always iterative, and all activities progress in parallel.

> Plan to iterate your analysis, design and implementation.

You won't get it right the first time, so integrate, validate and test as frequently as possible.

> Show your prototype early to the user. Andrew Begel: "After the first minute of watching somebody use my tool, I know a lot of things I have to fix".

Incremental Development

Plan to *incrementally* develop (i.e., prototype) the system

Proposed by Millis in 1980.

Plan to *incrementally* develop (i.e., prototype) the system.

> Always have a *running version of the system*,
> even if most functionality is yet to be implemented.
> *Integrate* new functionality as soon as possible.
> *Validate* incremental versions against user requirements.

Continuous integration is the keyword here.

“Continuous Deployment”

Nowadays we do not have the same model as 50 years ago. We do not ship software in boxes. The current approach to deployment has changed. Systems like GitHub and Facebook are notorious for continuously deploying new features.

How can you be sure that your small change will not interfere with other features?
- automatic testing
- testing with small groups of users then A/B testing

Roadmap

- Course Overview
- What is Software Engineering?
- Development Lifecycles
- **Software Development Activities**
- Methods and Methodologies

Requirements Collection

Requirements are informal, incomplete, and ambiguous

User requirements are often expressed informally:
- features
- usage scenarios

Although requirements may be documented in written form, they may be incomplete, ambiguous, or even incorrect.

“Verba volant, scripta manent”. Writing remains, and helps us thing.

Requirements Will Change

Validation is needed throughout system evolution

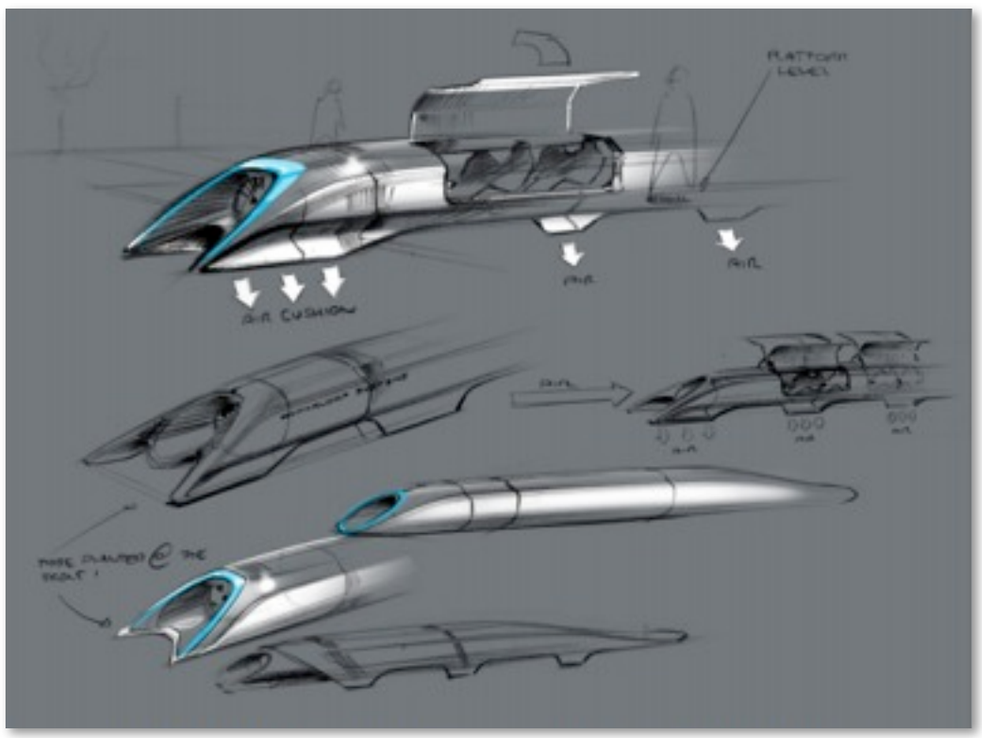
Requirements will change!

- inadequately captured or expressed in the first place
- user and business needs may change during the project

Validation is needed throughout the software lifecycle, not only when the “final system” is delivered!

- build constant feedback into your project plan
- plan for change
- early prototyping [e.g., UI] can help clarify requirements

Requirements Analysis and Specification



Analysis results in ...

a specification

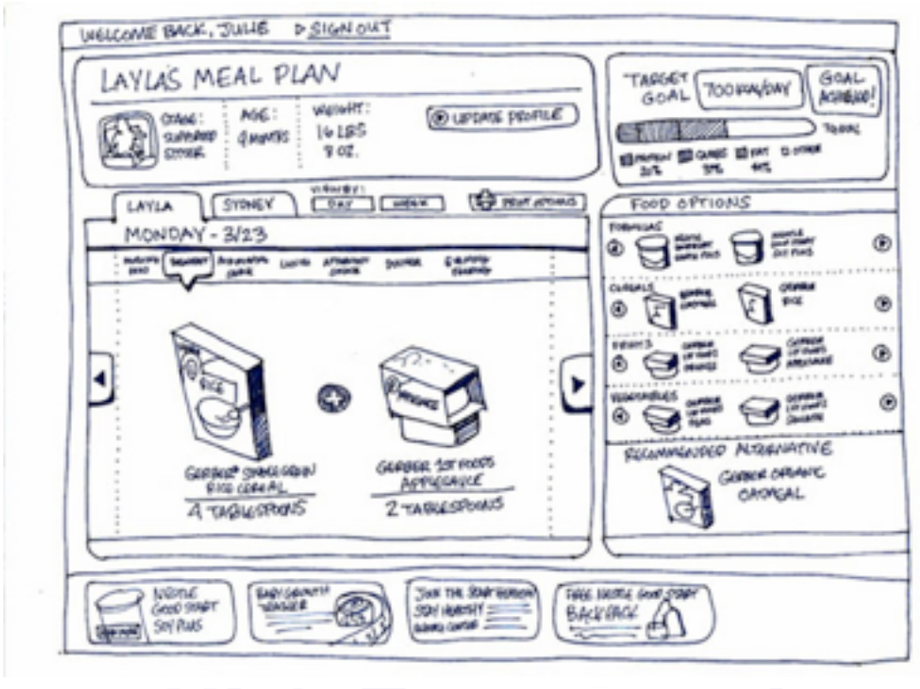
Analysis is the process of specifying what a system will do. The intention is to provide a clear understanding of what the system is about and what its underlying concepts are. A very good example of analysis is present in the Hyperloop documents leaked by Elon Musk. There one can see that the problem has been studied in detail, and a solution has been outlined only in order to conduct further discussions.

The result of analysis is a specification document.

Beware: what is not specified, remains at the attribution of the individual developer. And sometimes this is not what you want.

Specification is even more important when you develop software for yourself.

Prototyping



UI & Functional

Experimental

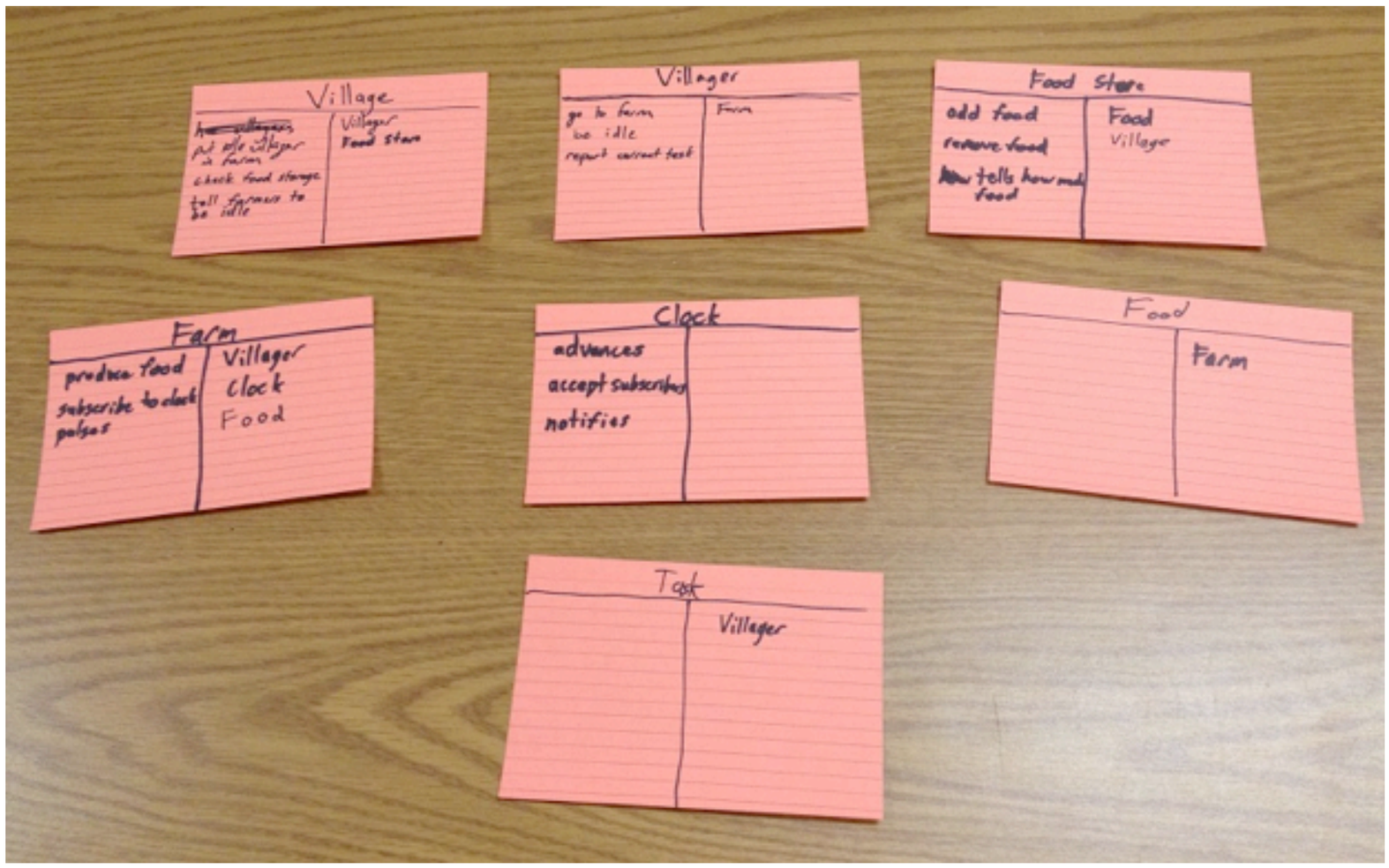
A prototype is a software program developed to test, explore or validate a hypothesis, i.e. to reduce risks.

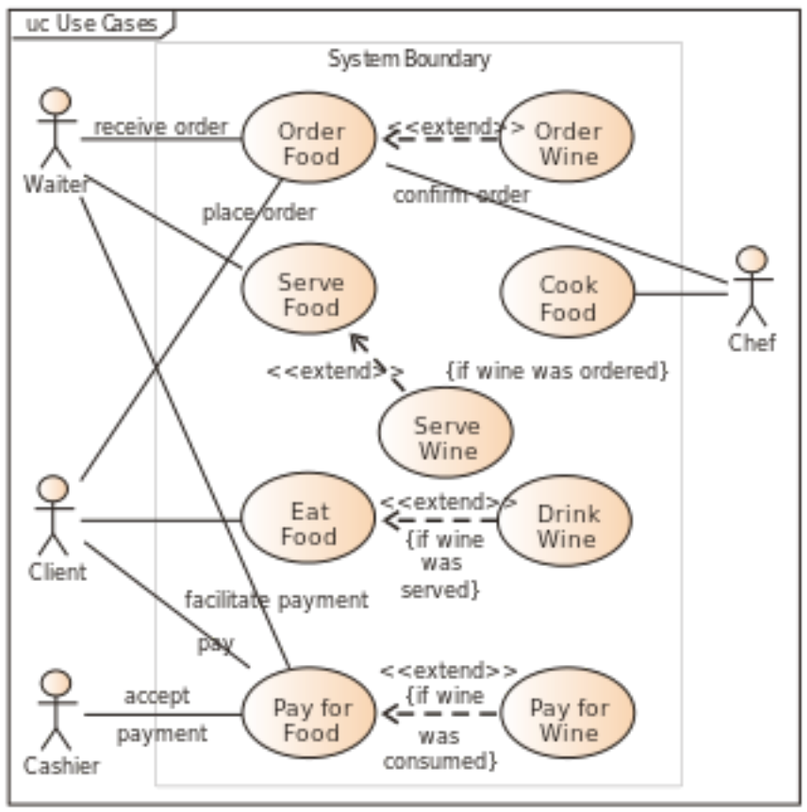
An exploratory prototype, also known as a throwaway prototype, is intended to validate requirements or explore design choices.

- UI prototype — validate user requirements
- rapid prototype — validate functional requirements
- experimental prototype — validate technical feasibility

Image: http://www.youtube.com/watch?v=TeS_U9qFg7Y

Object-Oriented Analysis and Design





An object-oriented analysis results in models of the system which describe:
- classes of objects that exist in the system
  - responsibilities of those classes
- relationships between those classes
- use cases and scenarios describing
  - operations that can be performed on the system
  - allowable sequences of those operations

Object-oriented design delivers models that describe:
- how system operations are implemented by interacting objects
- how classes refer to one another and how they are related by inheritance
- attributes and operations associated to classes

Image: http://paulgestwicki.blogspot.ch/2013/02/spring-2013-game-studio-modeling-first.html

Implementation and Testing

Implementation is the activity of constructing a software solution to the customer's requirements.

Testing is the process of validating that the solution meets the requirements.

The result of implementation and testing is a fully documented and validated solution.

Testing and implementation go hand-in-hand
Ideally, test case specification precedes design and implementation

The last 20% of the time takes 80% of the effort.

http://www.lifehacker.com.au/2009/02/before_and_after_barren_attic_to_programmers_paradise-2/
http://blog.treefortbikes.com/index.php/posts/1327

Maintenance

Maintenance is the process of changing a system after it has been deployed.

Repeatable, automated tests enable evolution and refactoring

Corrective maintenance: identifying and repairing *defects*
Adaptive maintenance: *adapting* the existing solution to new platforms
Perfective maintenance: implementing *new requirements*

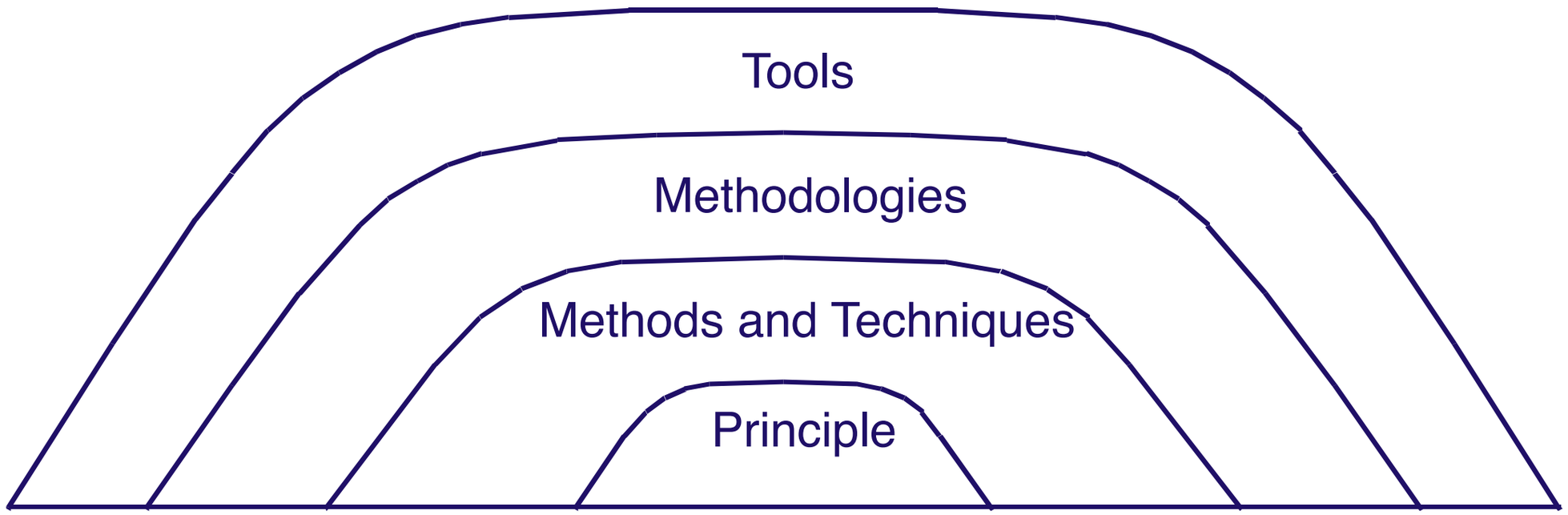
“Maintenance” entails:
- configuration and version management
- reengineering (redesigning and refactoring)
- updating all analysis, design and user documentation

Roadmap

- Course Overview
- What is Software Engineering?
- Development Lifecycles
- Software Development Activities
- **Methods and Methodologies**

Methods and Methodologies

Principle = general statement describing desirable properties
Method = general guidelines governing some activity
Technique = more technical and mechanical than method
Methodology = package of methods and techniques



— *Ghezzi et al. 1991*

Object-Oriented Methods: a brief history

- **First generation:**
  - Adaptation of existing notations (ER diagrams, state diagrams ...): Booch, OMT, Shlaer and Mellor, ...
  - Specialized design techniques:
    - *CRC cards; responsibility-driven design; design by contract*
- **Second generation:**
  - Fusion: Booch + OMT + CRC + formal methods
- **Third generation:**
  - Unified Modeling Language:
    - *uniform notation: Booch + OMT + Use Cases + ...*
    - *various UML-based methods (e.g. Catalysis)*
- **Agile methods:**
  - Extreme Programming
  - Test-Driven Development
  - Scrum …

What you should know!

- How does Software Engineering differ from programming or computer science?
- Why is the “waterfall” model unrealistic?
- Why plan to iterate? Why develop incrementally?
- Why is programming only a small part of the cost of a software project?

Can you answer these questions?

- Why do requirements change?
- How can you validate that an analysis model captures users' real needs?
- When can implementation start?

All images that are not otherwise attributed, are from wikipedia.