Introduction to software engineering

- The software crisis
- What is software engineering
- Software process lifecycle models
- Unified Software Development process
- Criteria of software quality

CE202 Software Engineering, Autumn term

Dr Cunjin Luo, School of Computer Science and Electronic Engineering, University of Essex

Computing in 1968

Apollo Guidance Computer. Culmination of years of work to reduce the size of the Apollo spacecraft computer from the size of seven refrigerators side-by-side to a compact unit weighing only 70 lbs

CICS (Customer Information Control System), an IBM transaction processing system, is released. Before CICS was introduced, many industries used punched card batch processing for high-volume customer transactions.

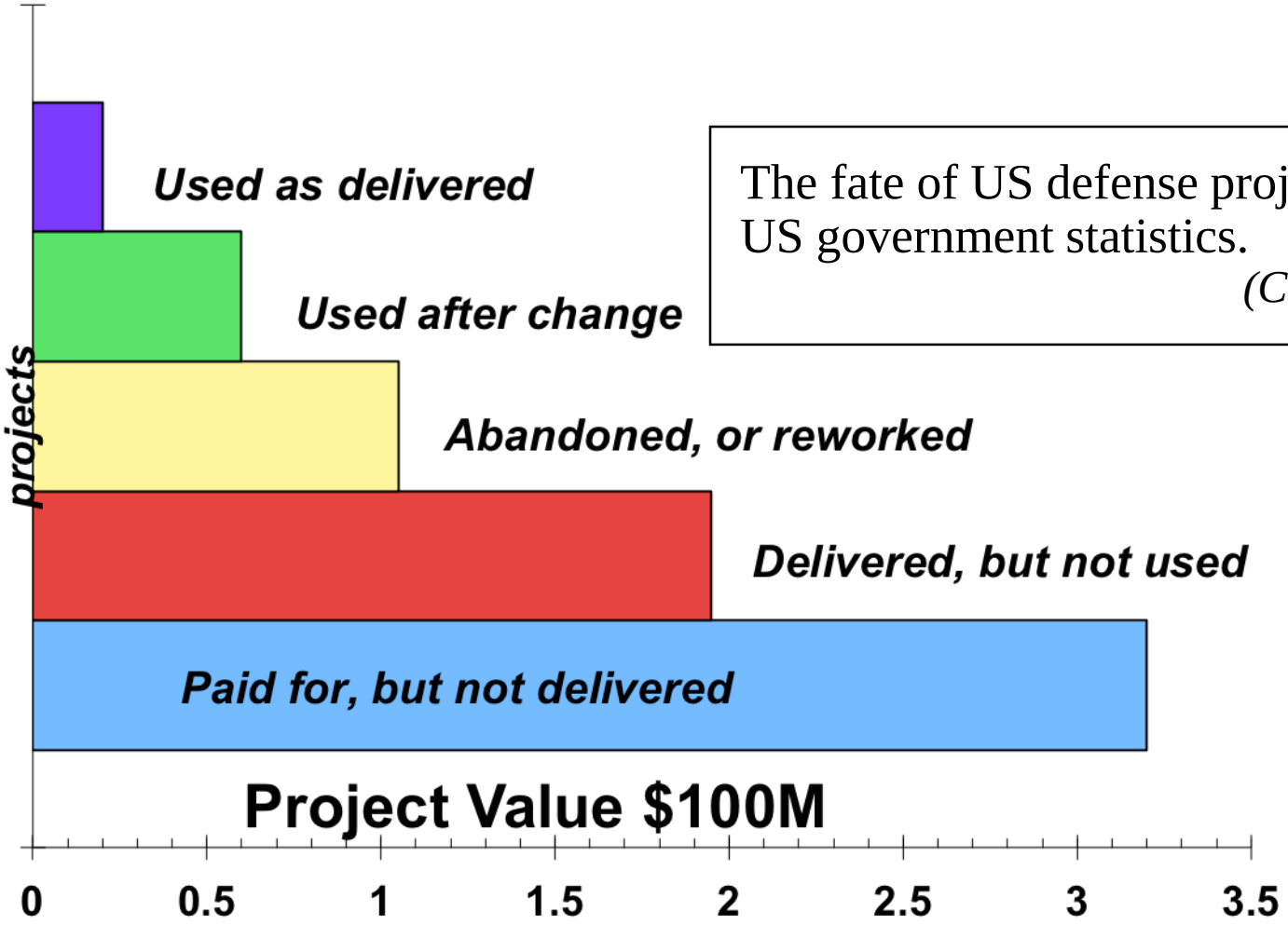
Data General designs the Nova minicomputer. It had 32 KB of memory and sold for $8,000.

Douglas Engelbart and his team at SRI, with funding from ARPA, unveil their experimental ‘oNline System’. It included collaborative editing, videoconferencing, word processing, and a strange pointing device jokingly referred to as a "mouse."

Software crisis 1968

- 'Software engineering' : "*The application of a systematic, disciplined, quantifiable approach to the development …and maintenance of software.*" [IEEE]
- NATO conference [Naur & Randell 69]
  - "The computer industry has a great deal of trouble in producing large and complex software systems."
  - **Problems**: Late deployment, budget overflows, unreliable and unsatisfactory systems, systems never delivered

Software Crisis 1989



The fate of US defense projects according to US government statistics.

(Connell & Shafer 1989)

Connell, John L. and Shafer, Linda, Structured Rapid Prototyping, Prentice-Hall, Inc., 1989.

[Carr 05] New York Times: 'Does Not Compute', http://www.nytimes.com/2005/01/22/opinion/22carr.html?pagewanted=print&position=

Software Crisis 2005

- "…software debacles are routine. And the more ambitious the project, the higher the odds of disappointment." [Carr 05]
- Only 34% of projects: timely & within budget.
- Some case studies:
  - FBI
    - Since 2001: a database on suspected terrorists
    - Jan. 2005: $M170, "not even close to having a working system"
  - Ford Motors
    - Since 2000, project "Everest": buying supplies, replacing legacy
    - Aug. 2004: $M200 over budget, abandoned (β-version slower than legacy)
  - McDonalds
    - Since 1999, project "Innovate", budget $M1,000
    - Killed in 2002, writing off $M170

Software Crisis 2008

- Ness settles arbitration case [Globes 29-Jan-2008]
  - Ness Technologies (Nasdaq: NSTC) has signed a settlement with Harel Insurance Investments (TASE: HARL), one of its clients.
  - The dispute developed in late 2006 when Harel [claimed] that Ness had breached the contract by not delivering the required software on time.
  - Ness challenged the claim, noting that Harel had dramatically increased the scope of the project since its start.
  - The dispute went to mandatory arbitration in January 2007, under the terms of the contract. Harel sought reimbursement for what it had paid as well as damages, totalling approximately $25 million.

http://www.globes.co.il/serveen/globes/docview.asp?did=1000303419&fid=1725

Software Crisis 2015

- The average success rate of software projects was **30.3%**, whereas **46%** of projects were challenged and **23.4%** of projects failed
- The average project time overrun rate was **76%**, project cost overrun was **52.5%** and the project feature delivery rate was **68.4%**
- A KPMG Survey, found that on average, about **70 %** of all IT-related projects fail to meet their objectives.
- Poorly defined applications (miscommunication between business and IT) contribute to a **66%** project failure rate, costing U.S. businesses at least **$30** billion every year.
  - 60% – 80% of project failures can be attributed directly to poor requirements gathering, analysis, and management.
  - 50% are rolled back out of production
  - 40% of problems are found by end users
  - 25% – 40% of all spending on projects is wasted as a result of re-work.
  - Up to 80% of budgets are consumed fixing self-inflicted problems

 From: 'Identifying the Reasons for Software Project Failure and Some of their Proposed Remedial through BRIDGE Process Models'. January 2015. INTERNATIONAL JOURNAL OF COMPUTER SCIENCES AND ENGINEERING 3(1):118-126, Ardhendu Mandal

Problems in Focus

- **Problem 1**: Unreliable products
  - Systems crash, halt, or demonstrate unexpected behaviour

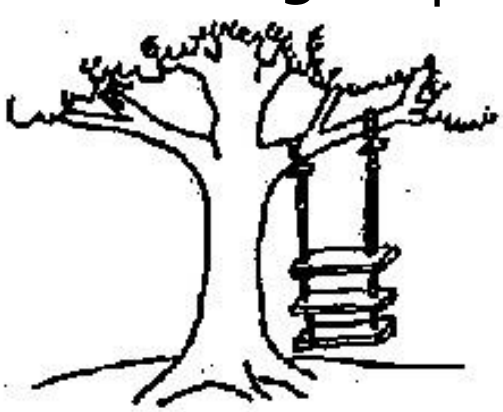
Software and Cathedrals are much the same: First we build them then we pray
--Sam Redwine, Jr.

- **Problem 2**: Maintenance is expensive
  - In reality, more than 90% (!) of the resources dedicated to software
  - Maintenance costs exceed initial estimate

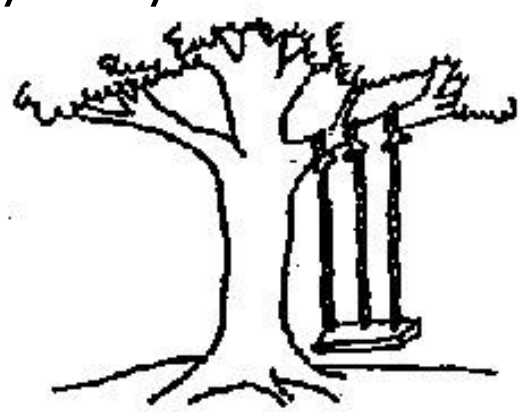
Samuel T. Redwine, Jr. - quote made at 4th International Software Process Workshop and published in Proceedings of the 4th International Software Process Workshop, Moretonhampstead, U.K., 11-13 May 1988.

Problems in Focus (Cont.)

- **Problem 3**: Deployed systems are "Incorrect"

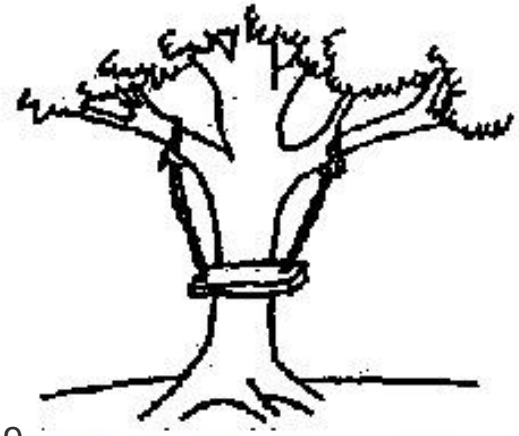


As proposed by the project sponsor.

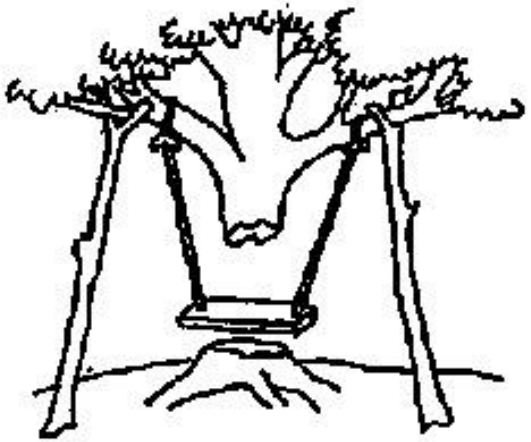


As specified in the project request.

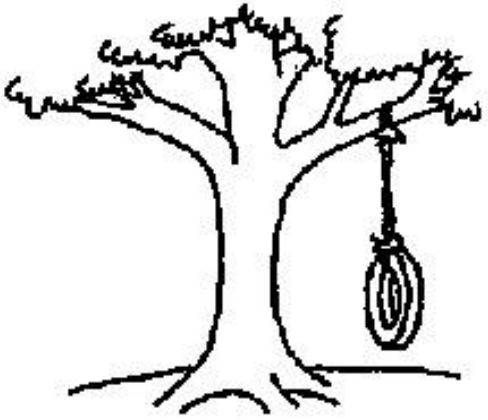
As designed by the senior analyst.



As produced by the programmers.

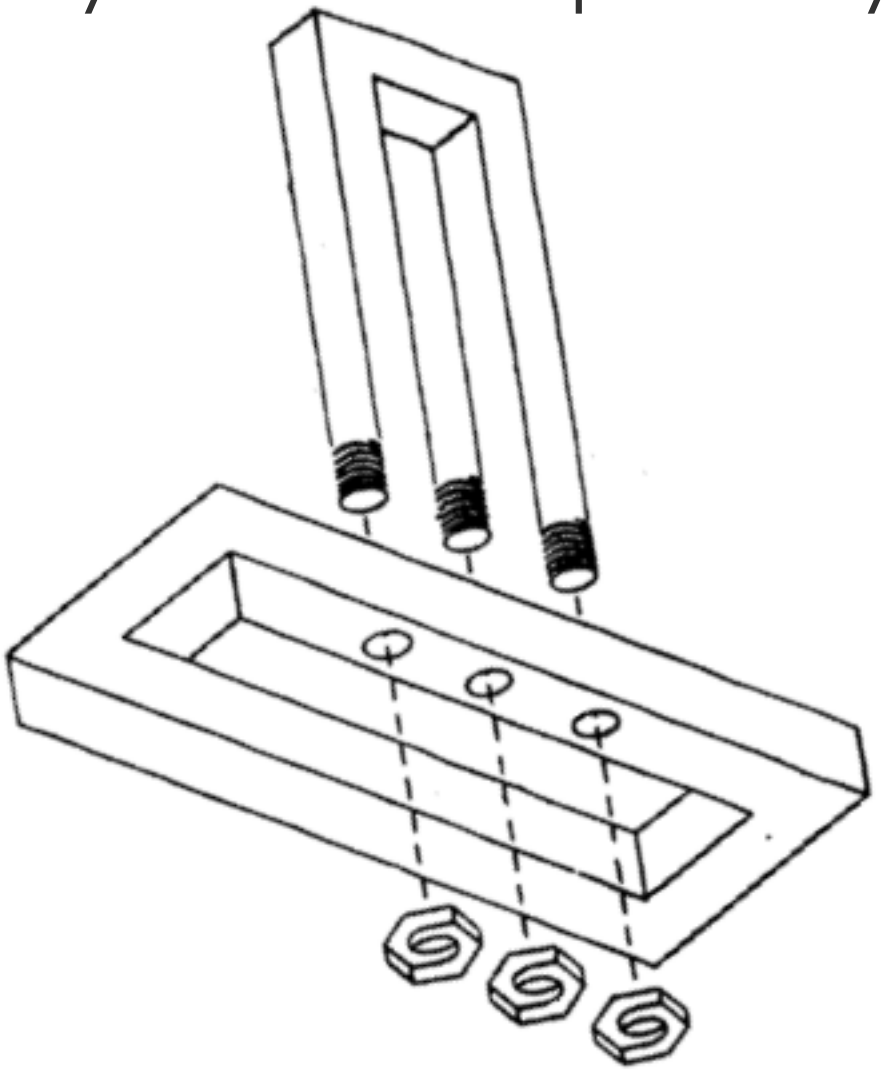


As installed at the user's site.



What the user wanted.

Can you develop this system?



Explanations to the (endless) crisis

- Software is very different from other manufactured (engineered) products
  - It is unlike cars, televisions, clothes, bridges, computers, …
- Underlying difficulties [Brooks 1987]:
  - Complexity
    - Software entities are more complex for their size than perhaps any other human construct
    - no two parts are alike
  - Changeability
    - Moore's law
    - Operational environment in constant flux: hardware, operating system, communication protocols, technologies, components, …
  - Invisibility
    - Bits are intangible & invisible

Brooks, Frederick P. (1987). *No Silver Bullet: Essence and Accidents of Software Engineering*. (Reprinted in the 1995 edition of *The Mythical Man-Month*)

What is software engineering?

- The application of engineering to software
- Branch of computer science dealing with software systems that are—
  - large and complex
  - built by teams
  - exist in many versions
  - last many years
  - undergo changes

Software engineering is a race between programmers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning.
-- Rich Cook, paraphrasing Einstein

Rich Cook – light fantasy author, USA, 'Wizardry' series of books

What is software engineering (cont.)

- Techniques (methods):
  - Formal procedures for producing results using some well-defined notation
  - Examples: functional/object decomposition, formal specification, interviews, creating class diagrams, prototyping, …
- Methodologies:
  - Collection of techniques applied across software development and unified by a philosophical approach
- Tools:
  - Instrument or automated systems to accomplish a technique
  - Examples: compiler (Sun Java), integrated development environment (Eclipse), configuration management (CVS servers), …
- Models
  - Class Diagrams, Interaction Diagrams, State machines, …

Software Engineering: A Working Definition

Software Engineering is a collection of **techniques**, **methodologies** and **tools** that help with the production of:

A high quality software system developed within a given *budget* before a given *deadline* while *change* occurs

Challenge: Dealing with complexity and change

Software Engineering: A Problem Solving Activity

- **Analysis:**
  - Understand the nature of the problem and break the problem into pieces
- **Synthesis/Design:**
  - Put the pieces together into a large structure

For problem solving we use techniques, methodologies and tools.

We will look at the difference between analysis and design during the course

Types of Software

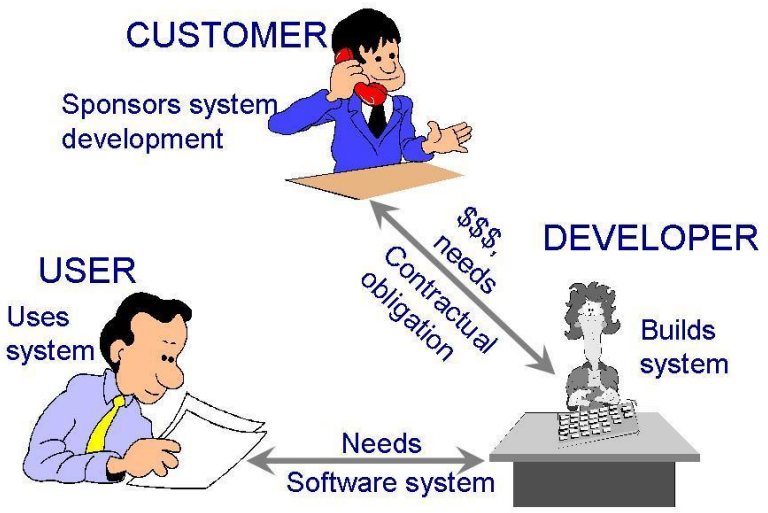
- Custom ("bespoke") software
  - one particular customer
  - developed "in-house" or outsourced
  - example: software developed by the IT division of a bank
- 'Commercial Off-The-Shelf' (COTS) or 'shrink-wrapped'
  - sold on the open market
  - can be cheaper and more reliable than custom software but might not fit the needs exactly
  - examples: email clients, image processors, word processors, ...
- Embedded software
  - runs on specific hardware devices
  - tied to specific hardware
  - examples: DVD player, microwave ovens, airplanes, weapons, ...

Characteristics of Systems

- Every system has:
  - Inputs and outputs
  - A purpose (related to transformation)
  - A boundary and an environment
  - Subsystems and interfaces
  - Control using feedback and feed-forward
  - Some emergent property
    - Example given by Ed Yourdon was the analyst who turned to a colleague and said "you are certainly more than the sum of your parts – you are an idiot!" The point being that if you look at the parts that make up a person, it would be difficult to find idiocy at any level of analysis other than a holistic view of the whole person.

Stakeholders in software project teams

- **Requirement analysts**: work with the customers to identify and document the requirements
- **Designers**: generate a system-level description of what the system is supposed to do
- **Programmers**: write lines of code to implement the design
- **Testers**: catch faults
- **Trainers**: show users how to use the system
- **Maintenance team**: fix faults that show up later
- **Librarians**: prepare and store documents such as software requirements
- **Configuration management team**: maintain correspondence among various artefacts

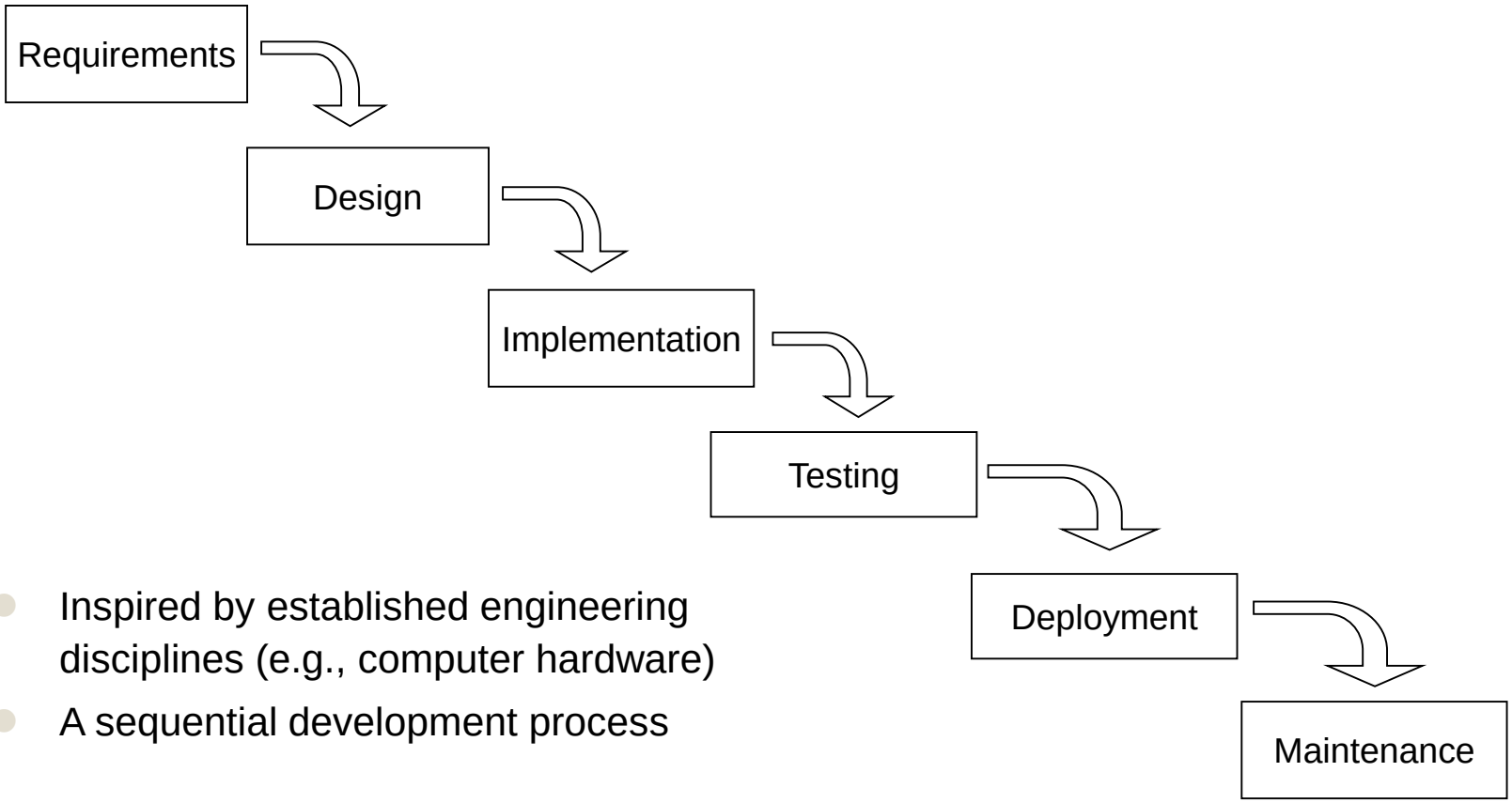


We will look at these different roles as we progress through the course

Software process lifecycle

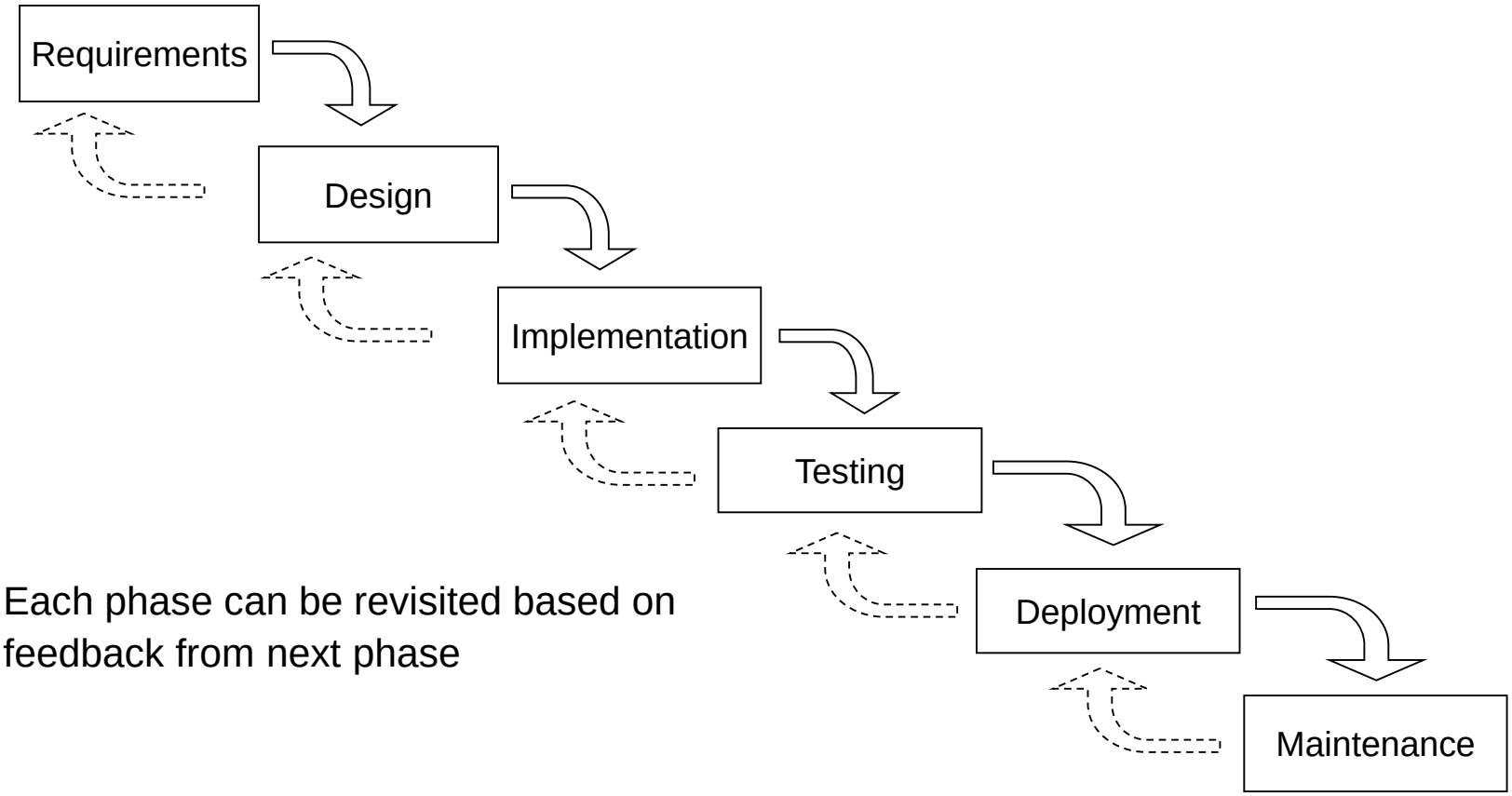
- Structures the software engineering project
- Determines:
  - Tasks: activities
  - Resources: personnel, equipment
  - Timing
  - Deliverables: requirements specification, system design, …
  - Tools
- Various models exist

Waterfall model



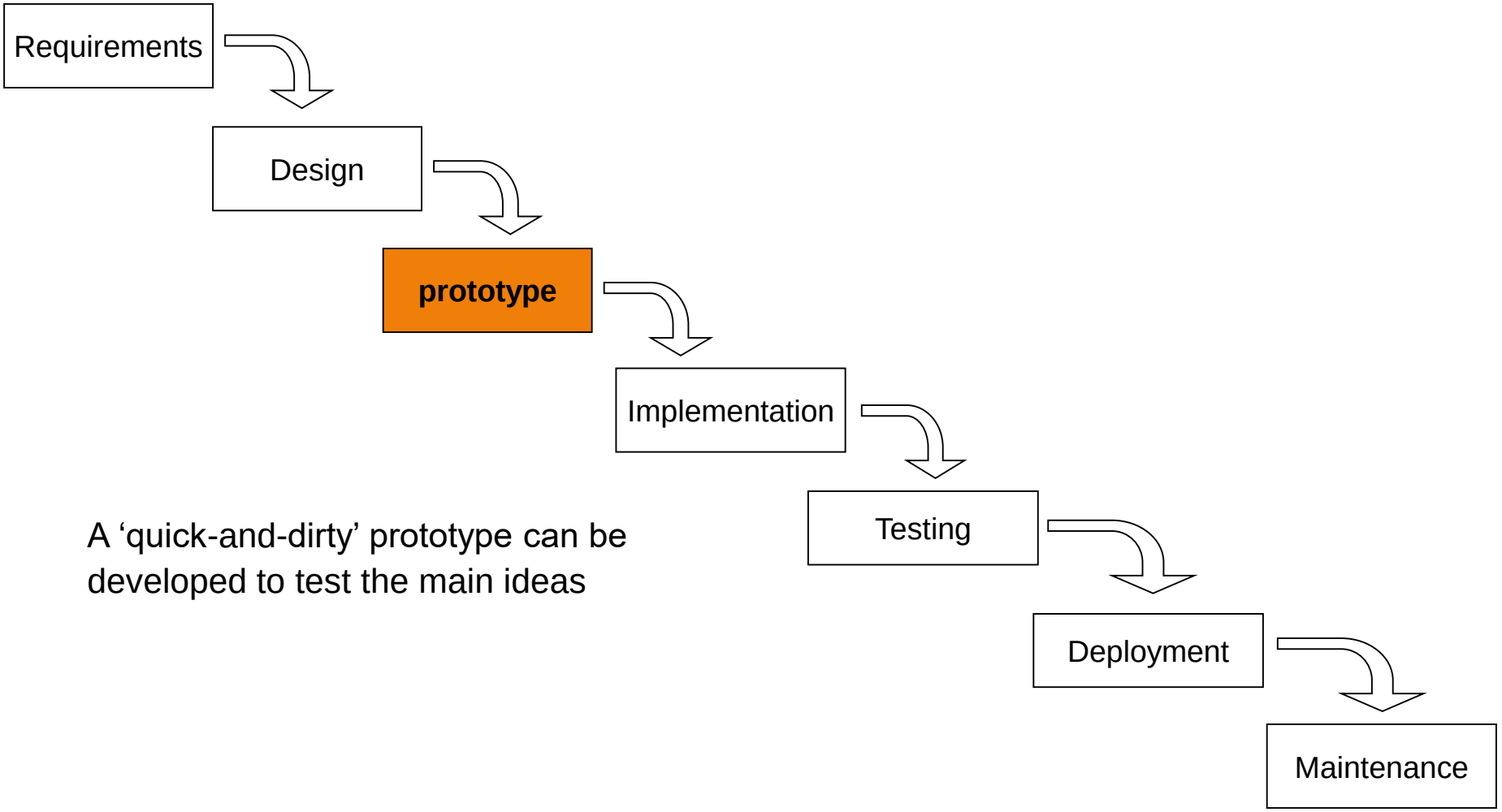
- Inspired by established engineering disciplines (e.g., computer hardware)
- A sequential development process

Waterfall with feedback



Each phase can be revisited based on feedback from next phase

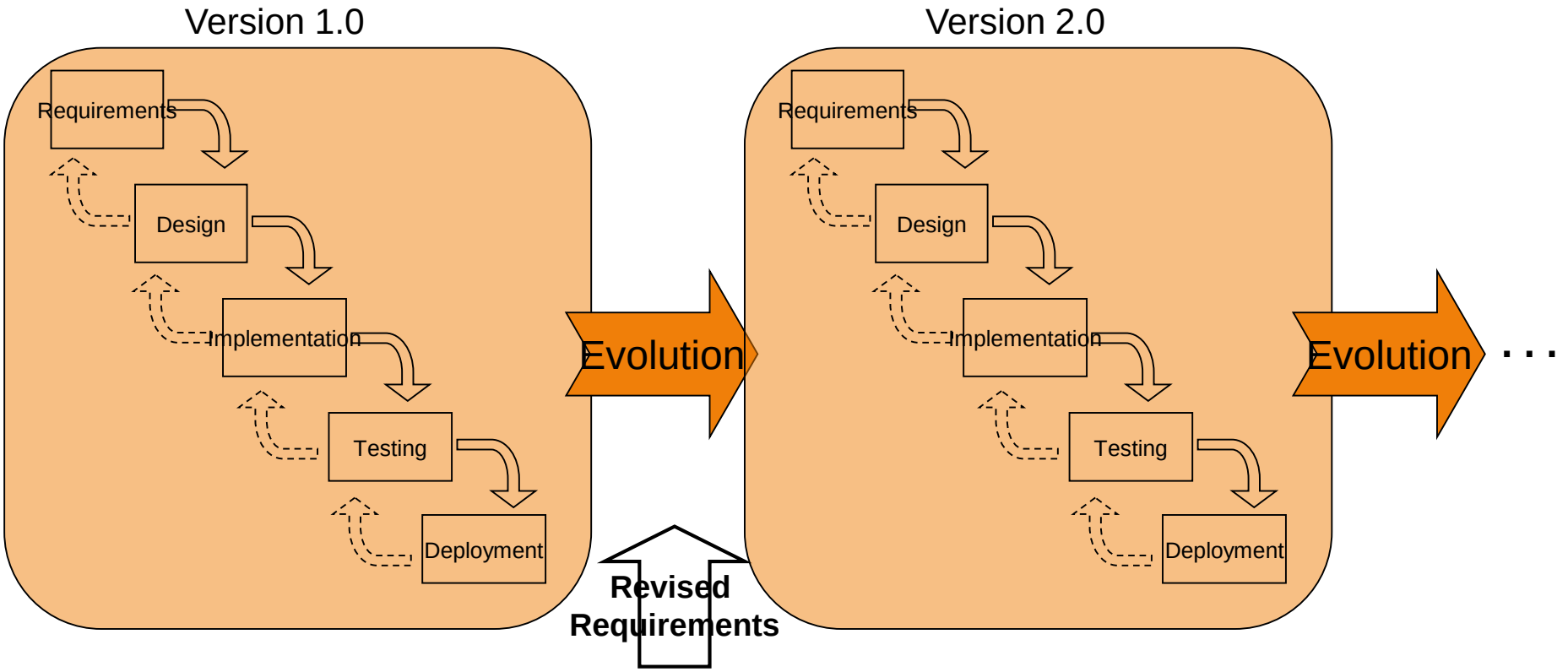
Waterfall with prototyping



A ‘quick-and-dirty’ prototype can be developed to test the main ideas

Evolutionary model

Evolution, not maintenance



Unified Software Development Process

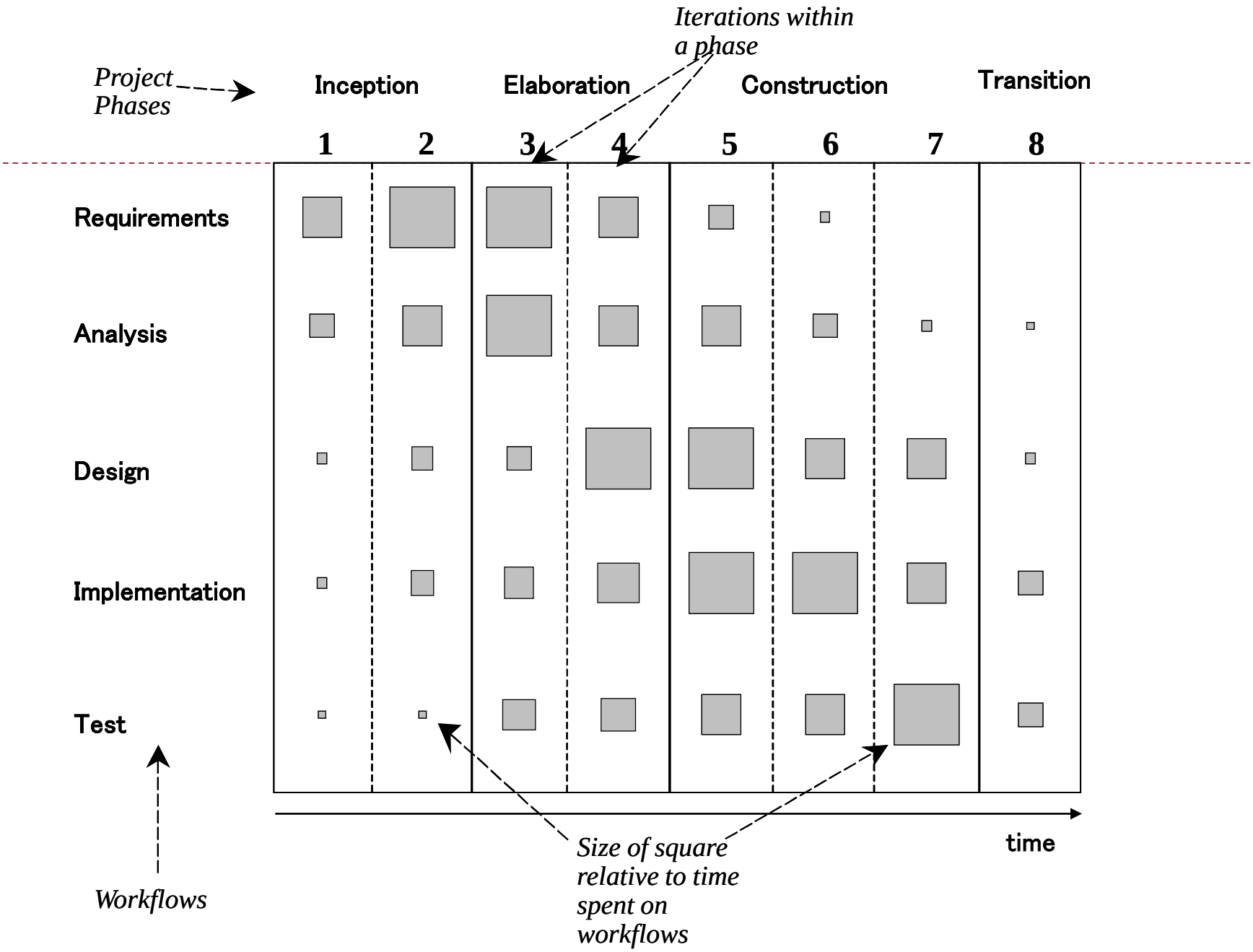
- Developed by the team that created UML
- Embodies best practice in system development
- Adopts an iterative approach with four main phases
- Different tasks are captured in a series of workflows

Four Phases

- Inception
- Elaboration
- Construction
- Transition

Phases, Workflows and Iterations

- Within each phase activities are grouped into workflows
- The balance of effort spent in each workflow varies from phase to phase
- Within phases there may be more than one iteration


Iterations within a phase
Project Phases
Inception
Elaboration
Construction
Transition
1
2
3
4
5
6
7
8
Requirements
Analysis
Design
Implementation
Test
time
Workflows
Size of square relative to time spent on workflows

Difference from Waterfall Life Cycle

- In a waterfall life cycle project the phases and the workflows are linked together
- In the Requirements phase, only Requirements workflow activities are carried out
- All Requirements activity should be completed before work starts on Analysis
- In an iterative life cycle project it is recognised that some Requirements work will be happening alongside Analysis work

For the Twelve facets of XP: see p86 Pfleeger

eXtreme or Agile programming

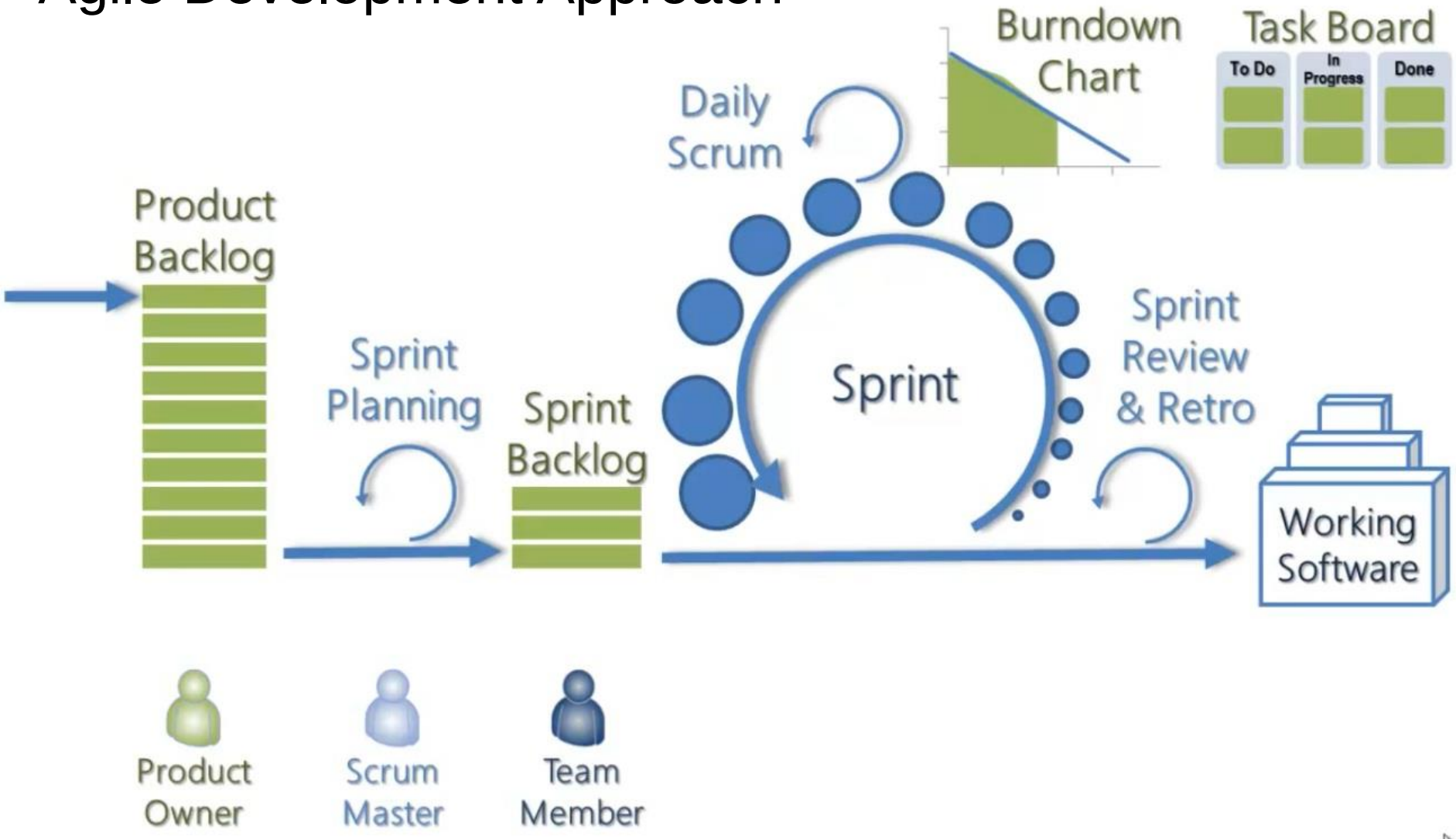
- The planning game (customer defines value)
- Small release
- Metaphor (common vision, common names)
- Simple design
- Writing tests first
- Refactoring
- Pair programming
- Collective ownership
- Continuous integration (small increments)
- Sustainable pace (40 hours/week)
- On-site customer
- Coding standard

Test Scenarios

The Agile approach believes that it is NOT possible to clearly define system requirements up front (as assumed by the waterfall approach).

Spike

Agile Development Approach



This is an iterative AND incremental development process

Metropolis Model

- Unstable resources
- Open teams
- Sufficient correctness
- Emergent behaviors
- Mashability
- Conflicting requirements
- Continuous evolution
- Focus on operations

The Metropolis Model: A New Logic for System Development – see http://www.slideshare.net/rickkazman/metropolis-model

Rick Kazman, University of Hawaii and Software Engineering Institute/CMU

Development methodologies

- We have only taken a brief look at:
  - Waterfall
  - Unified
  - Extreme & Agile
  - Metropolis
- There are others, for example:
  - Rapid Application Development (RAD)
  - Dynamic Systems Development Model (DSDM)
  - Lean Development (LD)
  - Etc, etc
- During this course we will focus on the waterfall approach, but these techniques can be universally applied

Waterfall Lifecycle and Main Outputs

Problem Definition
Viewpoints/Scope

Functional & Non-Functional requirements
Scenarios

Use-case diagrams
Use-case descriptions
Type Diagrams
Activity Diagrams
Prototypes
Class Diagram
Interaction Diagrams
State Machines

Prototypes
Class Diagram
Interaction Diagrams
Package diagrams
Architectural design

Coding
Unit testing
Version control

Integration testing
Unit testing

Deployment testing
User evaluation
Software evolution

All of these techniques can also be used in agile development

Finally: Criteria for software quality

- **External qualities** (directly visible to the client):
  - Correctness: Perform as intended by the client
  - Reliability: Absent from failures
    - Correct → Reliable
  - Robustness: Ability to survive failures and incorrect input
  - Efficiency (time/space): Affordable use of resources
  - Usability: Ease of learning & use
    - Determined largely by the human-computer interface (HCI)
    - Example: mobile phones
  - Safety: Does not pose a risk to humans & property
  - Secure: Vulnerability to malicious attacks

Criteria for software quality (cont.)

- **Internal qualities**: concern developers, not directly visible to clients
  - Maintainability: how well is the software designed, so as to make it
    - Flexible: can be evolved
    - Modular/ Loosely coupled: built from relatively independent modules
    - Cohesive: Maximize functional cohesion within modules
    - Comprehensible
  - Reusability: modules of it can be reused for related projects
  - Portability: can be adapted to run on different types of machines, operating systems
- We will particularly look at these issues in the **requirements** and **testing** phases of the course

History of UML

- Came out of the Unified Software Development Process (USDP) – Jacobson et al 1999
  - Built upon previous approaches
    - OO Software Engineering Method, Use cases (Jacobson 1992)
    - OO Design (Booch, 1994)
    - OO Analysis (OMT, Rumbaugh, 1991)
  - UML 1.1 1997
  - UML 2.0 2005
  - Latest release UML 2.5 (2015)
  - UML is not a development method by itself
  - Can be used by methods such as IBM Rational Unified Process (RUP)

Class Homework this week

- Look at the additional notes below
- Take a look at the SCRUM Methodology
  - See https://www.agilevideos.com/videoscategory/intro-to-scrum-videos/
  - Complete the 'Introduction to SCRUM' training package on that page
- Complete the quiz at the end of the Introduction

"The Scrum methodology of agile software development marks a dramatic departure from waterfall management. In fact, Scrum and other agile processes were inspired by its shortcomings. The Scrum methodology emphasizes communication and collaboration, functioning software, and the flexibility to adapt to emerging business realities — all attributes that suffer in the rigidly ordered waterfall paradigm."

In this lecture, we set the scene:

- Problems with creating software and why more rigour is needed (ie. software engineering)
- What is software engineering?
- Types of software and stakeholders involved
- Software development processes/methodologies
- Criteria for software quality
- History of UML

CE202 next steps

- Lecture next week
  - Overview of UML diagrams
  - UML Activity Diagrams
  - Requirements engineering & Use-case analysis
- Remember to do the SCRUM training homework!

Further reading …

- Chapter 5 and 21 of Bennett, McRobb and Farmer includes more about the Unified Process as well as Agile alternatives
- NATO Software Engineering conference 1968
  - http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF
- Software crisis 1989:
  - Connell, John L. and Shafer, Linda, Structured Rapid Prototyping, Prentice-Hall, Inc., 1989.
- Software crisis 2005:
  - New York Times: 'Does Not Compute', http://www.nytimes.com/2005/01/22/opinion/22carr.html?pagewanted=print&position=
- Software crisis 2008:
  - http://www.globes.co.il/serveen/globes/docview.asp?did=1000303419&fid=1725
- Brooks, Frederick P. (1987). *No Silver Bullet: Essence and Accidents of Software Engineering*. (Reprinted in the 1995 edition of *The Mythical Man-Month*)
- eXtreme programming: Twelve facets of XP: p86 Pfleeger
- Metropolis model: ACM July 2009 article - http://dl.acm.org/citation.cfm?id=1538808
- Jacobson, I., Booch, G., and Rumbaugh, J., *The Unified Software Development Process*, Addison-Wesley, ACM Press, 1999.