An Introduction to Software Engineering

Topics covered

- FAQs about software engineering
- Professional and ethical responsibility

Engineering

Engineering is the discipline and profession of applying technical and scientific knowledge and utilizing natural laws and physical resources in order to design and implement materials, structures, machines, devices, systems, and processes that safely realize a desired objective and meet specified criteria

Software Engineerring (SE)

- (Almost) everything depends on software.
- SE: theories, methods and tools for software development.
- No physical constraints (materials, physical laws)
  - simplifies: no limits on potential
  - makes things very complex!

Differences from other Engg. disciplines

- Some differences[1]
  - software: flexible, designed to change
  - (no mass production) build only one product, then copy
  - almost no fundamental laws apply
  - specifications can always change

[1]http://www.ibm.com/developerworks/rational/library/dec05/pollice/index.html

"...30% of all software projects are canceled, nearly half come in over budget, 60% are considered failures by the organizations that initiated them, and 9 out of 10 come in late."

— Economist magazine (Nov 27, 2004 p. 71)

Software Engineering (SE)

- Without SE, software is often:
  - unreliable, late delivered, over-budget
- Still: more art, less science
- No single ideal approach
  - but some fundamental ideas

FAQs about SE

- What is software?
- … software engineering?
- … the difference between SE and CS?
- … a software process?
- … a software process model?
- … the attributes of good software?
- … the key challenges facing software engineering?
- … software engineering methods?

What is software?

- Computer programs and
  - documentation: requirements, manuals
  - configuration data
- Software products may be:
  - Generic
  - Bespoke (custom)
  - line is becoming blurred

What is SE?

- Engineering discipline for all aspects of software production.
  - engineering:
    - theories, methods, tools
    - selectively
    - solutions for problems with no theories and methods
  - all aspects:
    - not just s/w development (“coding”)
- Choose the best technique based on current task

SE vs. CS

- CS: theory and fundamentals
- SE: practicalities of developing and delivering useful software.
- CS theories insufficient as a complete foundation for SE
  - (unlike e.g. physics and electrical engineering).

Software process

- Set of activities. Goal: production of software
  - Specification - expectations from system + development constraints
  - Development - production of the software system
  - Validation - checking that the software is what the customer wants
  - Evolution - changing the software in response to changing demands.

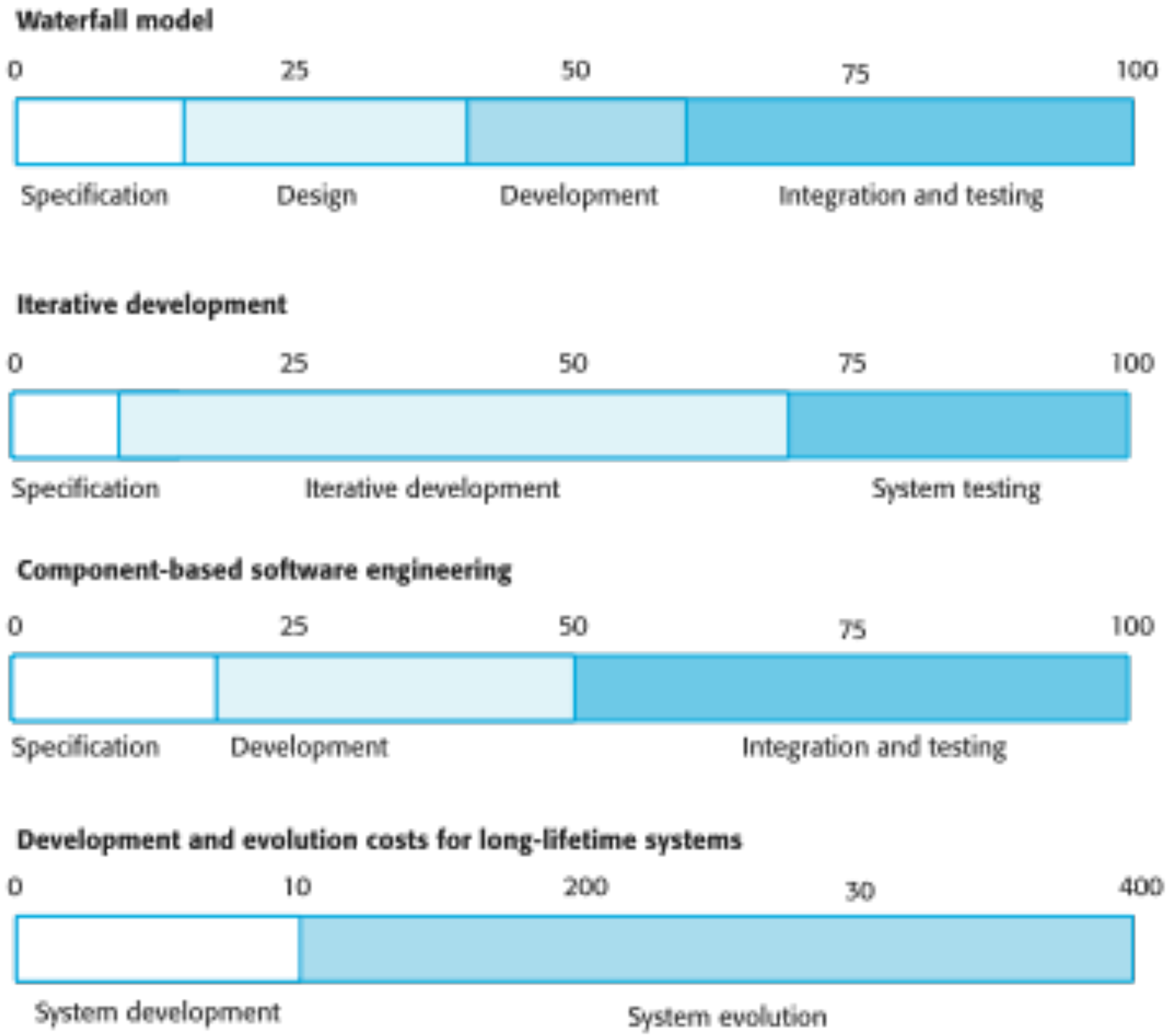
Software process model?

- Simplified representation of a software process
- Types of models:
  - Workflow - sequence of (human) activities
  - Data-flow - information flow
  - Role/action - who does what
- Generic process models
  - Waterfall
  - Iterative developmen
  - Component-based software engineering

Costs of SE

- 60%: development costs
- 40%: testing costs
- Distribution of costs depends on the development model that is used.

Activity cost distribution



Software Engineering methods

- Structured approaches to s/w development
  - how to structure programming
- Function-oriented (70s)
- Object-oriented (80s, 90s)
- UML-based unified approaches (currently)
- No single ideal method
  - e.g.: OO » great for interactive systems
    - not for real-time systems

Attributes of good software

- Maintainability
  - evolve to meet changing needs
- Dependability
  - trustworthy
- Efficiency
  - Efficient with use of system resources
- Acceptability
  - accepted by the users for which it was designed.
  - understandable, usable and compatible with other systems.

Key challenges facing SE

- Heterogeneity
  - Developing techniques for building software that can cope with heterogeneous platforms and execution environments
- Delivery
  - Developing techniques that lead to faster delivery of software
- Trust
  - Developing techniques that demonstrate that software can be trusted by its users.

Professional and ethical responsibility

- SE involves great responsibilities
  - more than application of technical skills
- Software engineers must behave in an honest and ethically responsible way
- Ethical behaviour is more than simply upholding the law.

Issues of professional responsibility

- Confidentiality
  - respect the confidentiality of their employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.
- Competence
  - Engineers should not misrepresent their level of competence.

Issues of professional responsibility

- Intellectual property rights
  - be aware of local laws governing the use of intellectual property such as patents, copyright, etc.
- Computer misuse
  - Software engineers should not use their technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial (game playing on an employer's machine, say) to extremely serious (dissemination of viruses).

ACM/IEEE Code of Ethics

- Code of ethical practice.
- Members sign up to the code of practice when they join.
- 8 Principles for practitioners, educators, managers, supervisors and policy makers, as well as trainees and students of the profession.

Ethical dilemmas

- Disagreement in principle with the policies of senior management.
- Your employer acts in an unethical way and releases a safety-critical system without finishing the testing of the system.
- Participation in the development of military weapons systems or nuclear systems.

Systems

What is a “system”?

A purposeful collection of interrelated components that work together to achieve some objective

Emergent System Properties

- present in complex systems
- “the whole is more than sum of its parts”
- can’t be atttributed to a single subsystem
- result from relationships/communication

Emergent System Properties

- *Functional:* when the parts work together for an objective
  - for e.g., the parts of a bicycle
- *Non-functional:* behavior in its operational environment
  - reliability
  - performance
  - safety
  - security
  - usability

Non-functional Properties

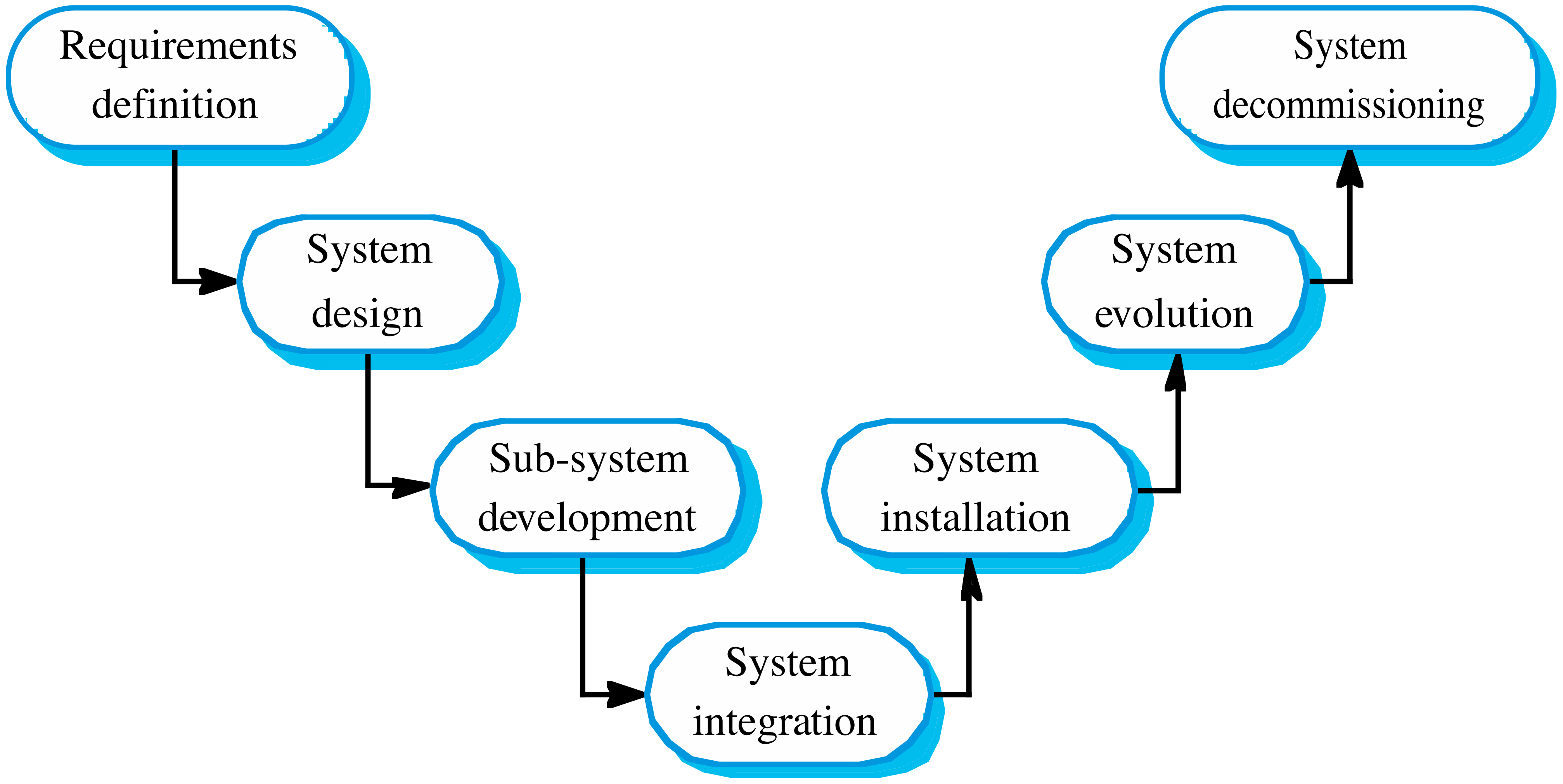
- Often more important than functional
- Must be considered at system level
  - not at component level
- E.g.: **reliability**
  - interdependent components
  - failure propogates

‘shall-not’ Properties

- Most properties are ‘shall’; easily measurable
  - *performance*
  - *accuracy*
- ‘shall-not’: properties not to be exhibited
  - *safety*: system shouldn’t behave in an unsafe way
  - *security*: system shouldn’t permit unauthorized access

Systems Engineering

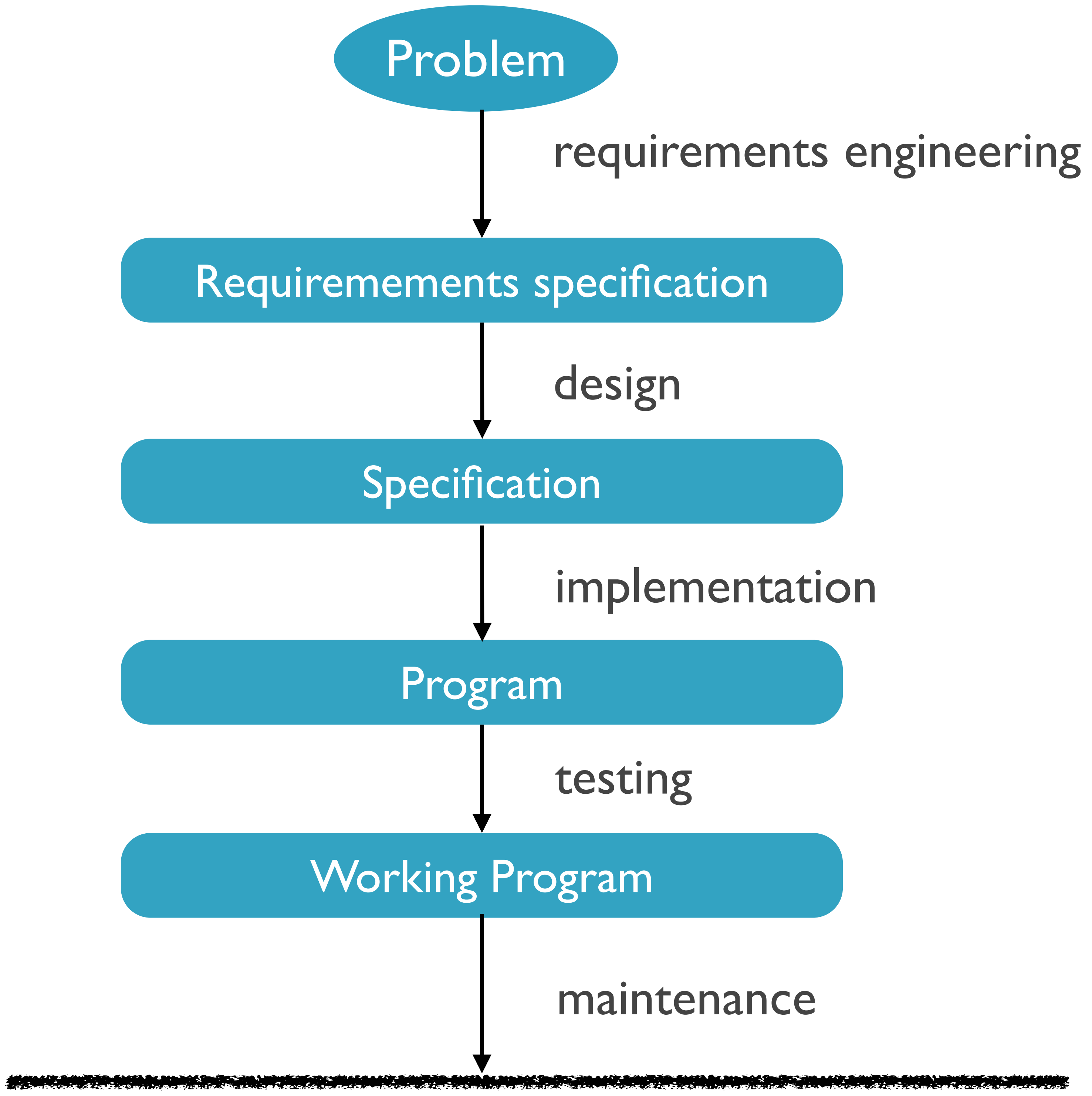
- Activities in the process of developing and maintaing a system



Software Engineering Process

- We will cover following phases in the process:
  - Specification/Requirements gathering
  - Design
  - Development
  - Evaluation
  - Evolution
- *Waterfall* or *Iterative*

The data-flow view



Requirements Engineering

- Complete *description* of problem
  - software’s functions
  - future extensions
  - documentation
  - response time & other performance
- Also, requirements posed by & on environment (the *constraints*)
  - hardware + software
  - number of users
- Result: **requirement specification**

Design / Modeling

- Create a model to be implemented
- Usually includes
  - *components*
  - *interfaces*
- Design decisions impact final quality
- Work more on *what* and less on *how*
- Result: **technical specification**
  - starting point for implementation

Implementation

- Focus on individual components
    - translate specification to modules, classes
- Usually includes another “design” phase:
    - pseudocode
- Focus **on:**
    - good documentation
    - flexible, easy to read code
    - reliable, correct
    - **Not on:** speed, features
- **Result:** executable program

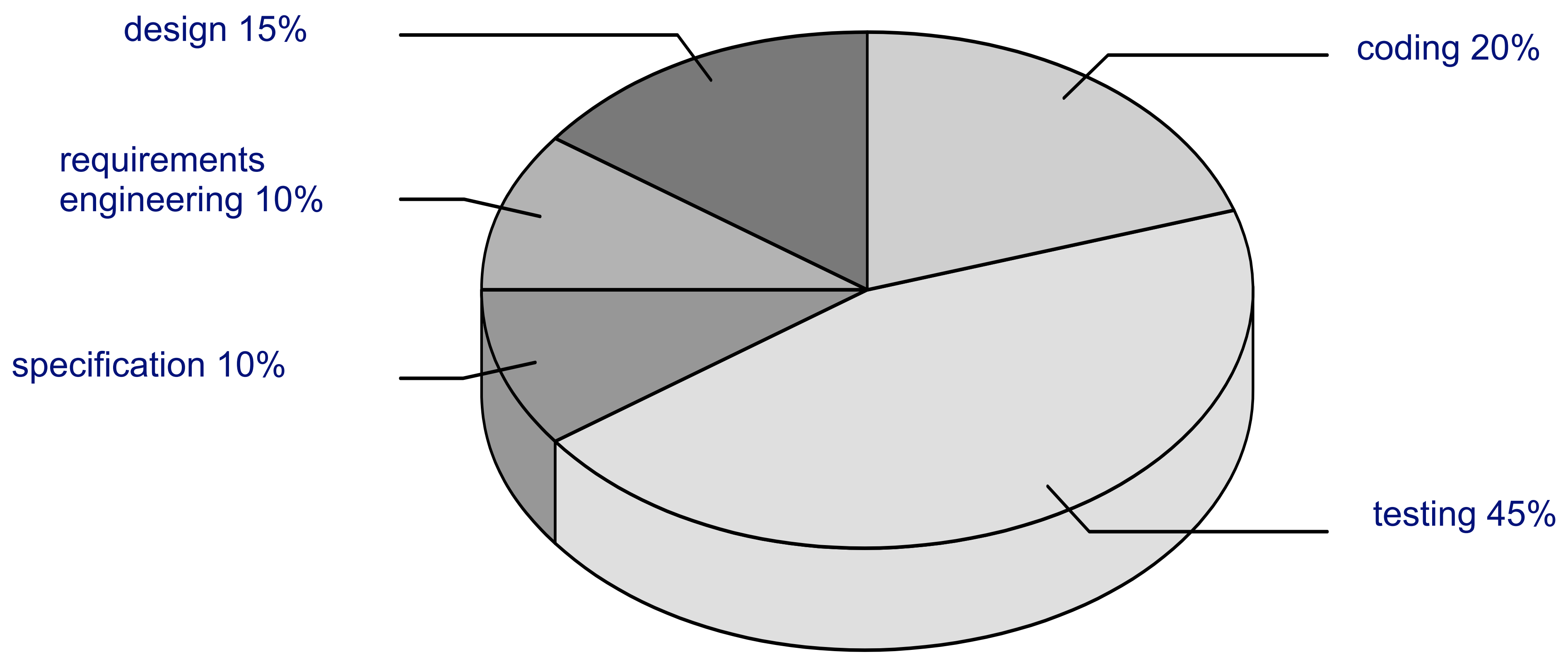
Testing

- Usually not just after implementation phase
- Start testing in first phase and refine
  - detect errors early and correct
- Testing at phase boundaries:
  - **verification:** transition is correct
  - **validation:** check against requirements

Maintenance

- After delivery
  - new errors
  - requrests for changes + enhancements
- Good maintenanability depends on earlier phases

Division of efforts



The 40-20-40 "rule"

Also 60-15-25 "rule"

Control

- Project control for
  - time
  - information
  - organization
  - quality
  - resources

Software Cycle* Revisited (Agile Development)

* process model

Traditional Software Lifecycle

- Phased development
  - identifiable milestones
  - usually correspond to some document's availability
- i.e. *Document-driven* development
  - *planning-driven, heavyweight*
  - traditional model (like *waterfall*)
- Good for large projects (more than 50 people)

Agile Lifecycles

- Agile methods, evolved from
  - Prototyping
  - Rapid Application Development
- Change is constant; specially requirements
- *lightweight*, *agile* process models
- develop quickly (prototype), get feedback, evolve
- involve users: people oriented
- development cycles: small & incremental
  - working system at end of each cycle

Agile Methods

- Key values:
  - individuals & interactions > processes & tools
  - working software > comprehensive documentation
  - responding to change > following a plan
- Usually a mixture of traditional & agile methods

Synchronize & stabilize

- Divide project in 4 - 5 builds
  - each adds functionalities
- *Synchronize:*
  - test and debug at end of day
  - executable version of software
- *Stabilize*: finalize a build
  - fix and freeze
  - acts as baseline for future development

Extreme Programming (XP)

- Pure agile method
  - Takes best practices to extreme
- For e.g., *pair programming*
- Simplest design, since future is unclear
  - complete a task, then improve (refactor)
  - «««« (pg 68)

Software Engineering Management

Introduction

- Software development projects involve
  - several people
  - long period of time
- Need for careful **planning** and control
- At a higher level than the process model

Steps in a Project Plan

- Choose process model
- Project organization: interactions, teams, roles
- Standards, guidelines
- Resources
- Quality assurance
- Budget and Schedule
- Changes
- Delivery

Control

- Project control for
  - time
  - information
  - organization
  - quality
  - resources