1. Overview

Why software engineering?

Key questions

- What exactly is software engineering?
- How is software engineering different from programming?
- Why do we need to learn software engineering?
- What are the main techniques used in software engineering?

Engineering

- Engineering is …
  - The application of scientific principles and methods
  - To the construction of useful structures & machines
- Examples
  - Mechanical engineering
  - Civil engineering
  - Chemical engineering
  - Electrical engineering
  - Nuclear engineering
  - Aeronautical engineering

Software Engineering

- The term is over 40 years old: NATO Conferences
  - Garmisch, Germany, October 7-11, 1968
  - Rome, Italy, October 27-31, 1969
- The reality is finally beginning to arrive
  - Computer science as the scientific basis
    - Other scientific bases?
  - Many aspects have been made systematic
    - Methods/methodologies/techniques
    - Languages
    - Tools
    - Processes

Software Engineering in a Nutshell

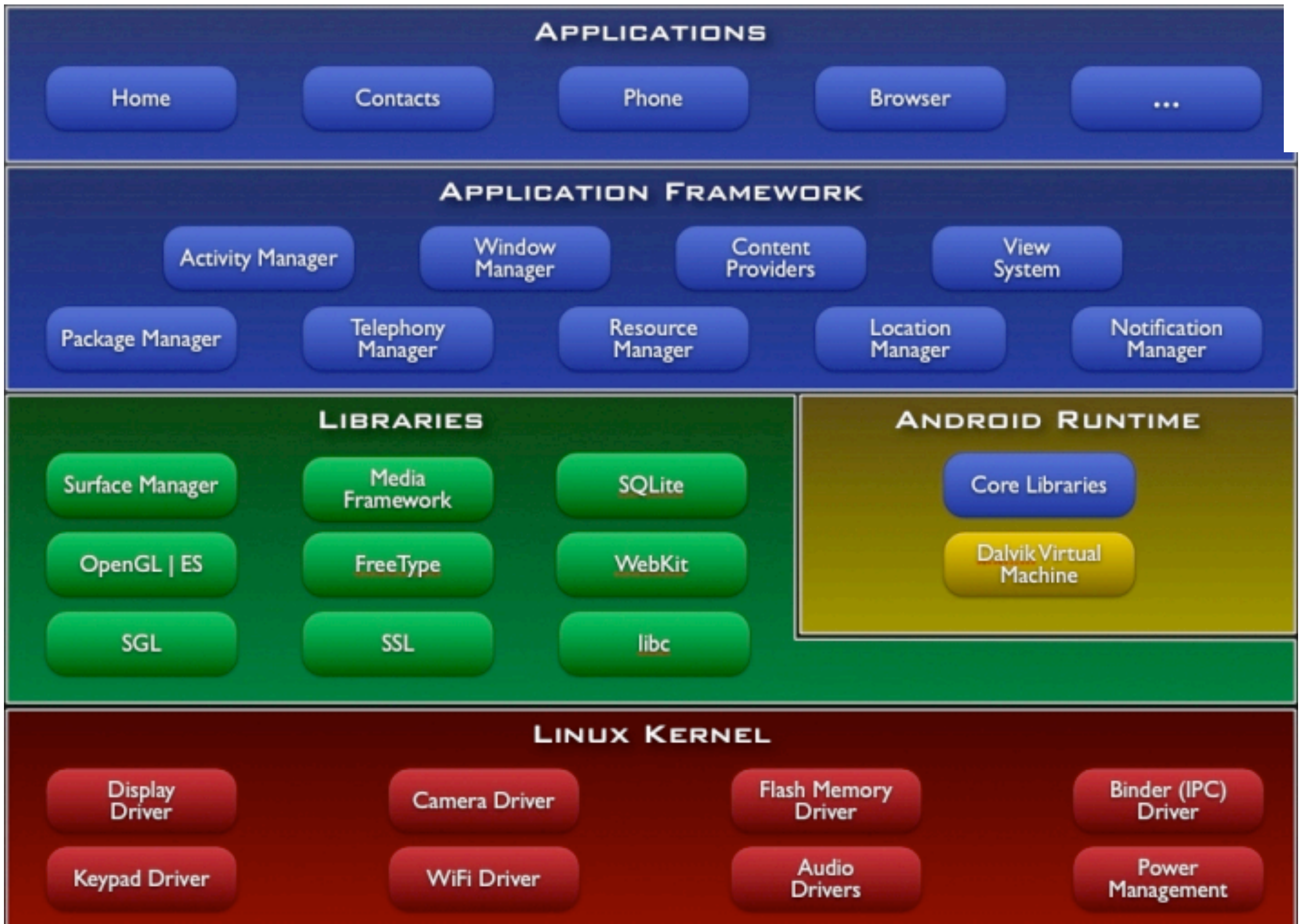
- Development of software systems → size/complexity warrants team(s) of engineers
  - multi-person construction of multi-version software [Parnas 1987]
- Scope
  - study of software process, development principles, techniques, and notations
- Goal
  - production of quality software, delivered on time, within budget, satisfying customers' requirements and users' needs

Software is Important

- Software is our lives
- Let us see some examples

Example: mobile phone

Source code is 2GB in size



Example: A380

- A380
- 1400 separate programs
- There is a software project just to manage all the software!
- Clearly safety-critical features of the software

But, software is difficult to get right …….

Software loses money

- The black swan effect (BBC news, Aug. 26, 2011)
  – One in six big IT projects go over-budget by an average of 200% (Oxford and Mckinsey)
  – Most of the project costs 30% more, one in six runs over 300% more
  – More details of the story: http://www.bbc.co.uk/news/technology-14677143

Software makes most expensive fireworks

- [Ariane 5 Maiden Flight]

Software cause misery

- [The 2003 North East Blackout]

Software kills

- London Ambulance Service
  - 1992, computerised ambulance despatch system fails

- Therac-25
  - 2 people died and several others exposed to dangerous levels of radiation because of software flaws in radiotherapy device

Danger is close to us

- Chinese Train Crash, July 23
  - Software error in signalling system

Ever-Present Difficulties

- Few guiding scientific principles
- Few universally applicable methods
- As much managerial / psychological / sociological as technological

Why These Difficulties?

- SE is a unique brand of engineering
  - Software is malleable
  - Software construction is human-intensive
  - Software is intangible
  - Software problems are unprecedentedly complex
  - Software directly depends upon the hardware
    - It is at the top of the system engineering "food chain"
  - Software solutions require unusual rigor

Software Engineering ≠ Software Programming

- Software programming
  - Single developer
  - “Toy” applications
  - Short lifespan
  - Single or few stakeholders
    - Architect = Developer = Manager = Tester = Customer = User
  - One-of-a-kind systems
  - Built from scratch
  - Minimal maintenance

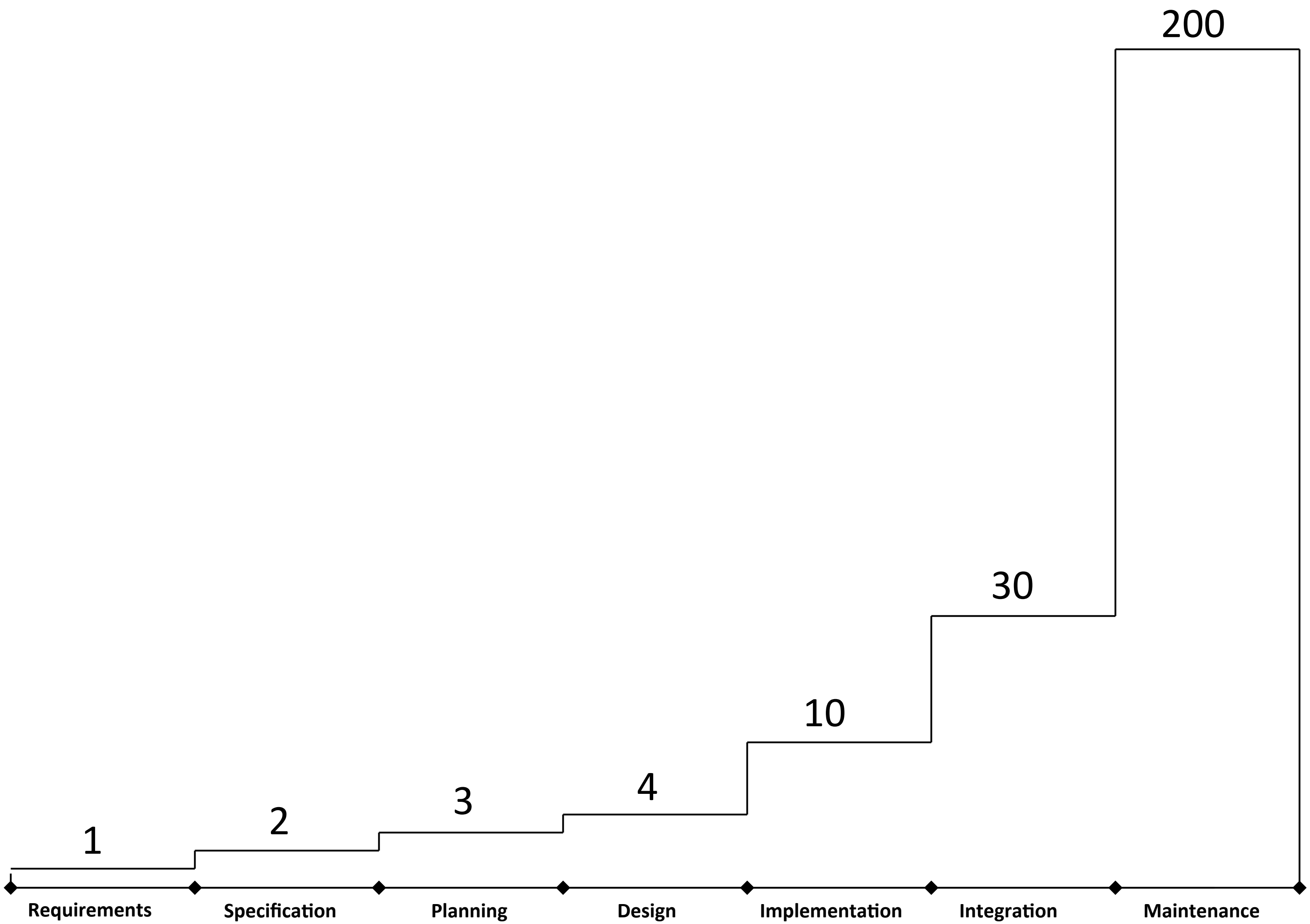
Software Engineering ≠ Software Programming

- Software engineering
  - Teams of developers with multiple roles
  - Complex systems
  - Indefinite lifespan
  - Numerous stakeholders
    - Architect ≠ Developer ≠ Manager ≠ Tester ≠ Customer ≠ User
  - System families
  - Reuse to amortize costs
  - Maintenance accounts for over 60% of overall development costs

Economic and Management Aspects of SE

- Software production = development + maintenance (evolution)
- Maintenance costs > 60% of all development costs
  – 20% corrective
  – 30% adaptive
  – 50% perfective
- Quicker development is not always preferable
  – higher up-front costs may defray downstream costs
  – poorly designed/implemented software is a critical cost factor

Relative Costs of Fixing Software Faults



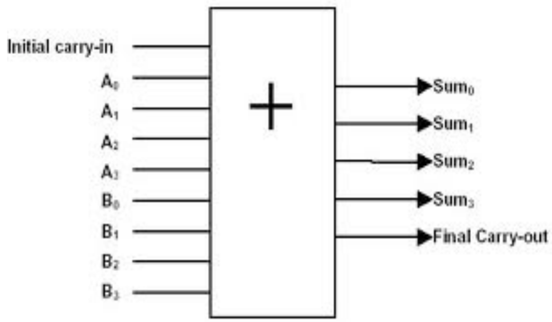
Mythical Man-Month by Fred Brooks

- Published in 1975, republished in 1995
  - Experience managing development of OS/360 in 1964-65
- Central argument
  - Large projects suffer management problems different in kind than small ones, due to division in labor
  - Critical need is the preservation of the conceptual integrity of the product itself
- Central conclusions
  - Conceptual integrity achieved through chief architect
  - Implementation achieved through well-managed effort
- Brooks' s Law
  - Adding personnel to a late project makes it later

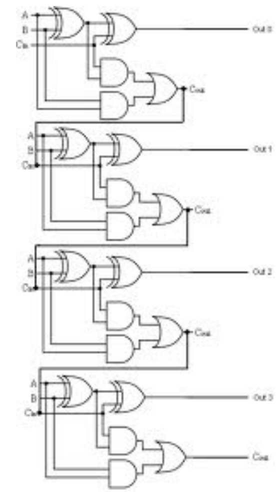
Abstraction and Modularity

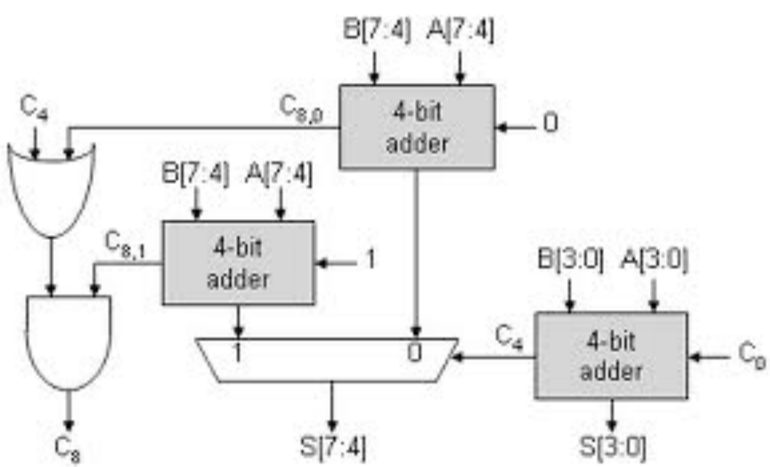
- Abstraction
  - Simplification of complex things by omitting undesired details
  - Simplification allows us to reason (calculate)
- Modularity
  - Isolation a reusable piece of functionality
  - Provide input/output and hide how the output is computed

Modularity

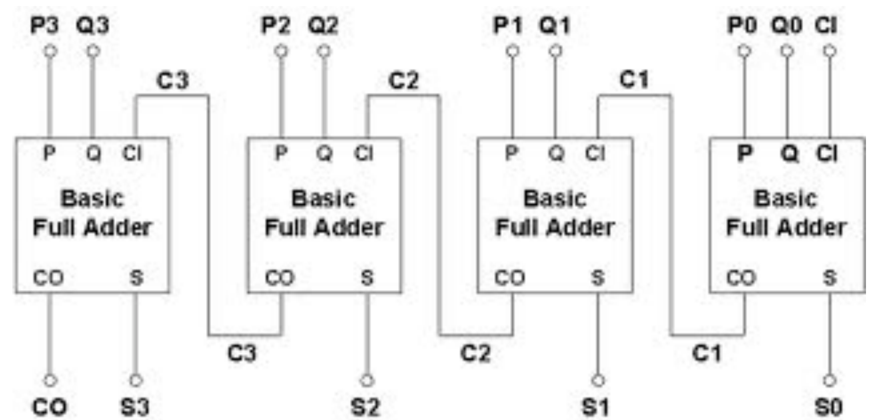


Initial carry-in
A_0
A_1
A_2
A_3
B_0
B_1
B_2
B_3
+
Sum_0
Sum_1
Sum_2
Sum_3
Final Carry-out





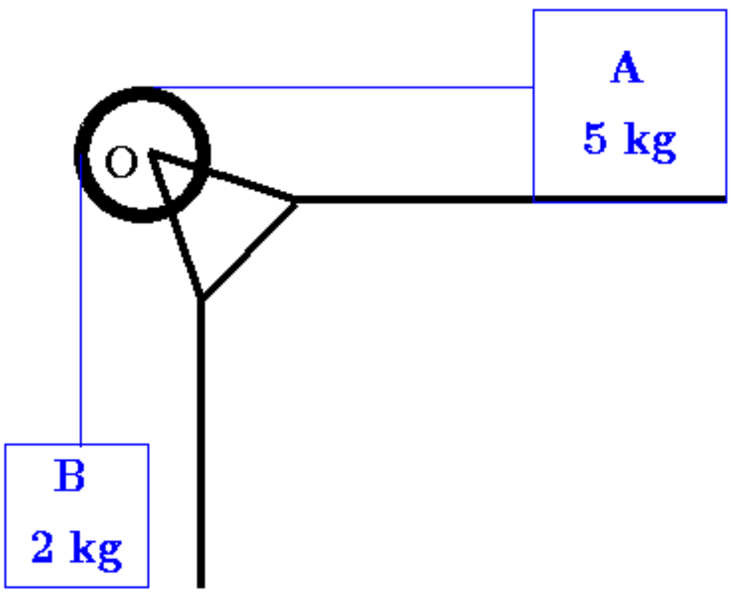
B[7:4] A[7:4]
C_4
$C_{8,0}$
4-bit adder
0
B[7:4] A[7:4]
$C_{8,1}$
4-bit adder
1
B[3:0] A[3:0]
C_4
4-bit adder
C_0
1
0
C_8
S[7:4]
S[3:0]

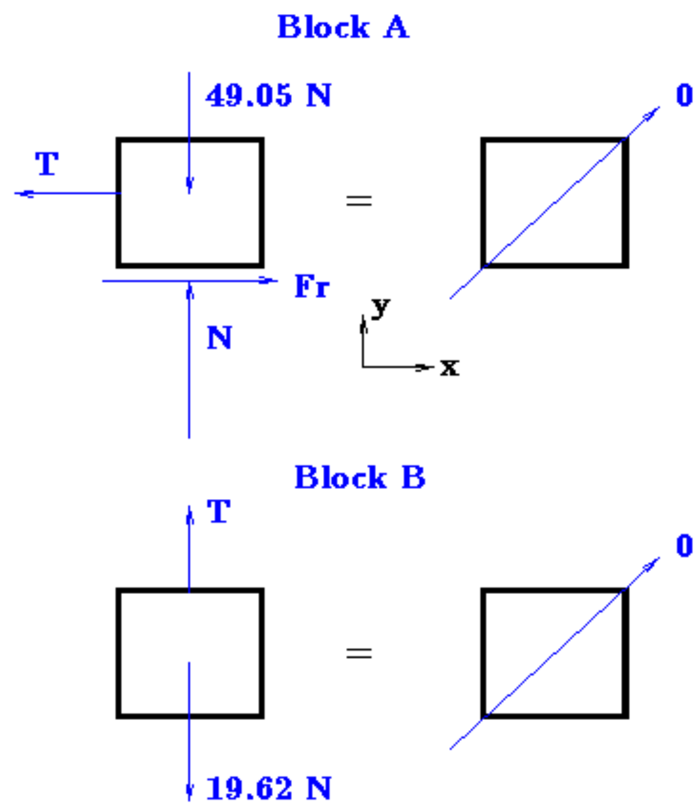


P3 Q3
P2 Q2
P1 Q1
P0 Q0 CI
C3
C2
C1
P Q CI
Basic Full Adder
CO S
C3
C2
C1
CO S3
S2
S1
S0

Abstraction: free-body diagram

A pulley system





So what is software engineering?

- Main activities
  - Know what we are going to build (Requirements)
  - Figure out a way to divide and conquer (Analysis)
  - Write good code (Design and Imlemetation)
  - Putting things back together (Integration)
  - Verify if we build the right thing (Testing)
  - Make changes without breaking anything (CM)
- Follow a process
  - When and where to perform the activities
  - Repeat good things in the next project

Requirements

- Problem Definition → Requirements Specification
  - determine exactly what the customer and user want
  - develop a contract with the customer
  - specifies what the software product is to do
- Difficulties
  - client asks for wrong product
  - client is computer/software illiterate
  - specifications are ambiguous, inconsistent, incomplete

Architecture/Design

- Requirements Specification → Architecture/Design
  - architecture: decompose software into modules with interfaces
  - design: develop module specifications (algorithms, data types)
  - maintain a record of design decisions and traceability
  - specifies how the software product is to do its tasks
- Difficulties
  - miscommunication between module designers
  - design may be inconsistent, incomplete, ambiguous

Implementation & Integration

- Design → Implementation
  - implement modules; verify that they meet their specifications
  - combine modules according to the design
  - specifies how the software product does its tasks
- Difficulties
  - module interaction errors
  - order of integration may influence quality and productivity

Component-based Development

- Third-party software “pieces”
- Plug-ins / add-ins
- Applets
- Frameworks
- Open Systems
- Distributed object infrastructures
- Compound documents
- Legacy systems

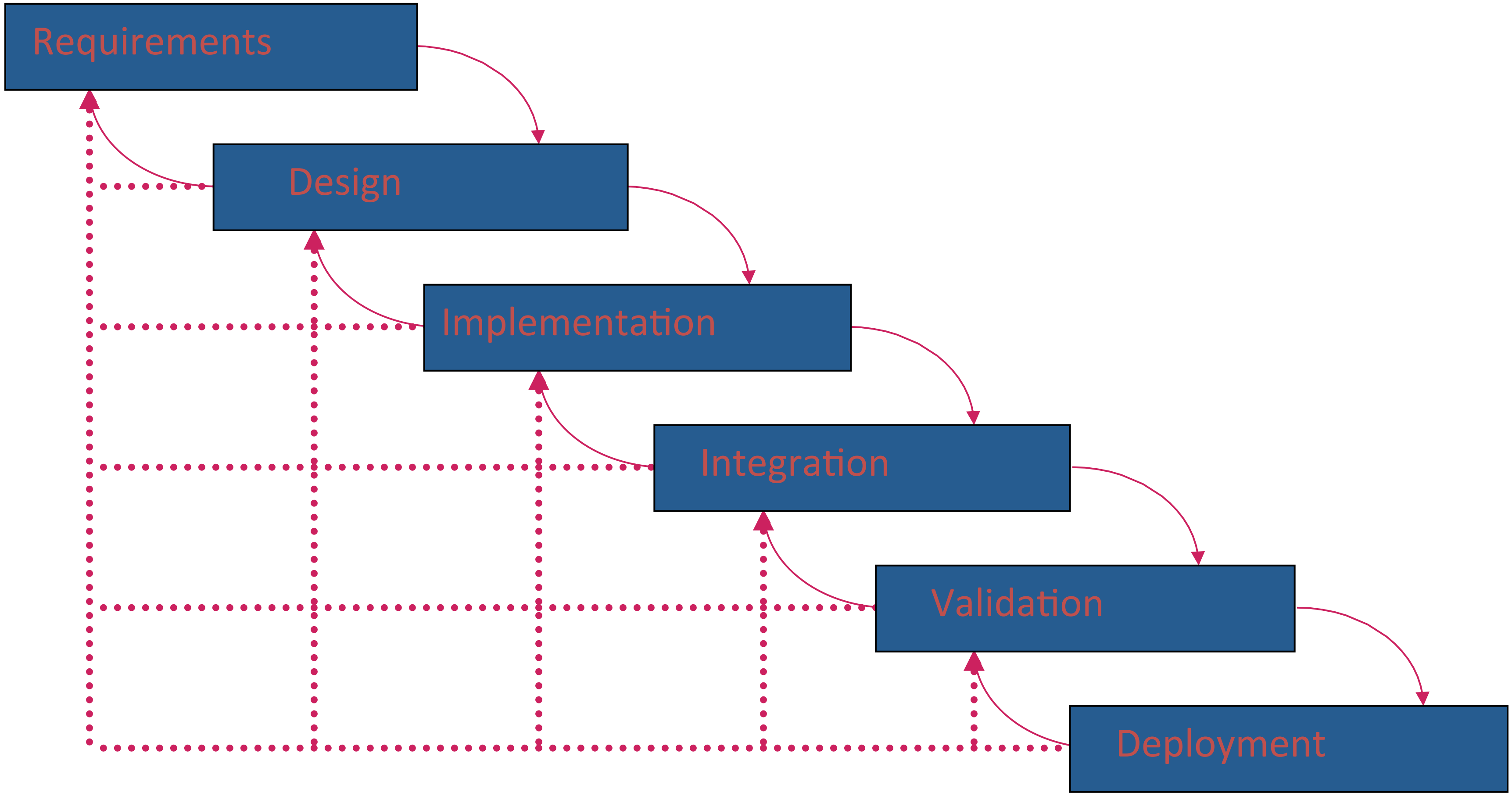
Verification and Validation

- Analysis
  - Formal verification
  - Informal reviews and walkthroughs
- Testing
  - Dynamic
  - "Engineering"
  - White box vs. black box
  - Structural vs. behavioral
  - Issues of test adequacy

Deployment & Evolution

- Operation → Change
  - maintain software during/after user operation
  - determine whether the product still functions correctly
- Difficulties
  - rigid design
  - lack of documentation
  - personnel turnover

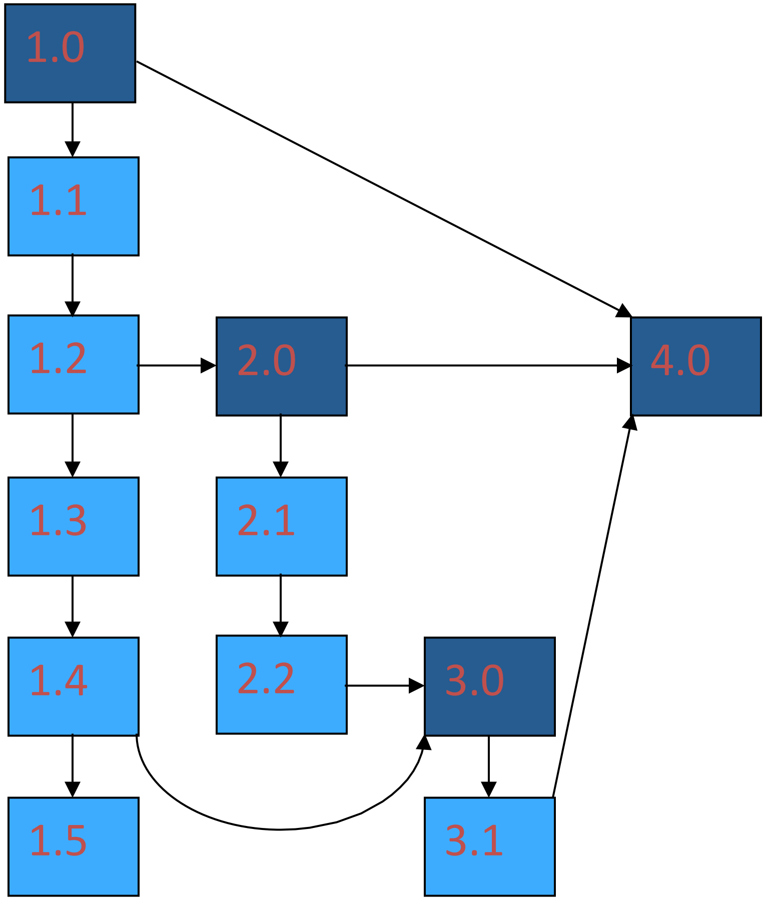
Software Development Lifecycle
Waterfall Model



Configuration Management (CM) [Tichy 1988]

- CM is a discipline whose goal is to control changes to large software through the functions of
  - Component identification
  - Change tracking
  - Version selection and baselining
  - Software manufacture
  - Managing simultaneous updates (team work)

CM in Action



Firefox RELEASES

Firefox release notes are specific to each version of the application. Select your version from the list below to see the release notes for it.

6.0
5.0.1
5.0
4.0
3.6
3.5.7
3.5.6
3.5.5
3.5.4
3.5.3
3.5.2
3.5.1
3.5
3.0.17
3.0.16
3.0.15
3.0.14
3.0.13
3.0.12
3.0.11
3.0.10
3.0.9

2.0.0.20
2.0.0.19
2.0.0.18
2.0.0.17
2.0.0.16
2.0.0.15
2.0.0.14
2.0.0.13
2.0.0.12
2.0.0.11
2.0.0.10
2.0.0.9
2.0.0.8
2.0.0.7
2.0.0.6
2.0.0.5
2.0.0.4
2.0.0.3
2.0.0.2
2.0.0.1
2.0

1.5.0.12
1.5.0.11
1.5.0.10
1.5.0.9
1.5.0.8
1.5.0.7
1.5.0.6
1.5.0.5
1.5.0.4
1.5.0.3
1.5.0.2
1.5.0.1
1.5
1.0.8
1.0.7
1.0.6
1.0.5
1.0.4
1.0.3
1.0.2
1.0.1
1.0

0.9.3
0.9.1/0.9.2
0.9
0.8
0.7.1
0.7
0.6.1
0.6
0.5
0.4
0.3
0.2
0.1

Software Qualities

- Qualities (a.k.a. "ilities") are *goals* in the practice of software engineering
- External vs. Internal qualities
- Product vs. Process qualities

External vs. Internal Qualities

- External qualities are visible to the user
  - reliability, efficiency, usability
- Internal qualities are the concern of developers
  - they help developers achieve external qualities
  - verifiability, maintainability, extensibility, evolvability, adaptability

Product vs. Process Qualities

- Product qualities concern the developed artifacts
  - acceptability, maintainability, understandability, performance
- Process qualities deal with the development activity
  - products are developed through process
  - maintainability, productivity, timeliness

Conclusion

- What exactly is software engineering?
  - Large software systems require many people and long term maintenance
- How is software engineering different from programming?
  - Team work vs. Individual work
- Why do we need to learn software engineering?
  - Our lives depend on software
- What are the main techniques used in software engineering?
  - Abstraction and modularity and others