CS471 Lecture 01

Software Engineering Introduction and Motivation
Sommerville Ch2

What is Engineering?

What is Engineering?

- "the application of mathematics, science, economics, empirical evidence, etc. to invent, innovate, design, build, maintain, research, and improve structures, machines, tools, systems, components, materials, processes, solutions, and organizations. "

What is Software Engineering?

Software Engineering Definitions

- "…an engineering discipline that is concerned with all aspects of software production from initial conception to operation and maintenance"

  -Sommerville

- "…the application of a systematic, disciplined, and quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software…"

  -IEEE

Software Engineering

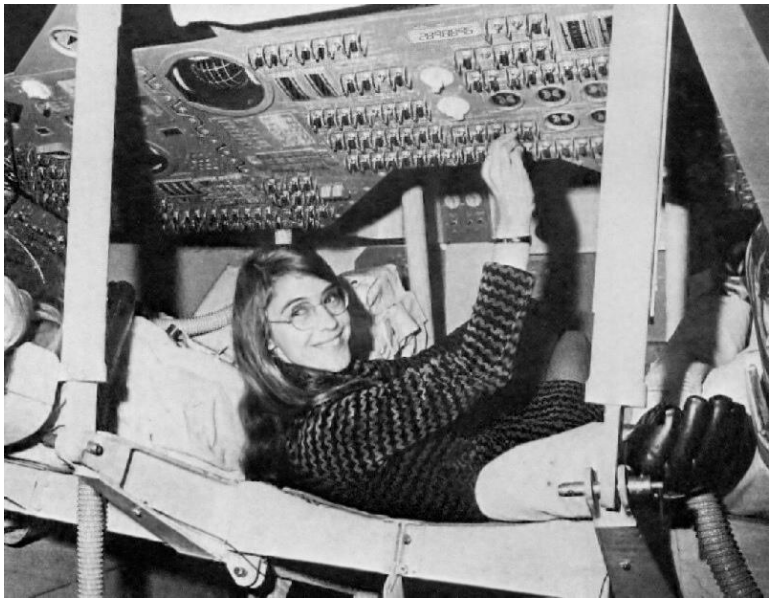
- First software (early 50's)
  - cost of hardware dominates
  - programs seem to be less important

Software Engineering

- First software (early 50's)
  - cost of hardware dominates
  - programs seem to be less important
- Software crisis (late 60's)
  - hardware becomes cheaper
  - custom software becomes complex and expensive
  - software production lags behind the need
  - software engineering discipline is born (early 70's)

Nascent Software Engineering Applications: Aerospace

- Margaret Hamilton, Lead Flight Software Designer, Apollo Program
- Prevented an abort of the Apollo 11 lunar landing



Nascent Software Engineering Applications: Aerospace

- See source code at:

https://github.com/chrislgarry/Apollo-11

- 80KLOC (i.e., 80,000 Lines of Code) written in Assembly

Original Software Engineering Objectives

- Improve the following competing resources
  - Quality
  - Schedule
  - Cost
- Largely focused on the development of large aerospace and enterprise applications

Questions about software

- Why does it take so long to get software completed?
- Why are costs so high?
- Why can't all errors be found before the software is put into production?
- Why is it difficult to measure the progress at which software is being developed?

High-level Explanations to Questions about software

- Software is developed (or engineered), not "manufactured" (in the classical sense)
- Software does not "wear out" (as do traditional concrete products), but it "deteriorates" during requirements, design, development, maintenance
- Most software is custom built rather than assembled from existing components

Software Engineering vs. [other] Engineering

- One of the essential technologies of today
  - essential for economy
  - essential for security
- Technology of the same importance as
  - mechanical engineering
  - electrical engineering, etc.
- How does software and engineering differs from other engineering fields?

Software Engineering vs. [other] Engineering

- Other branches of engineering use standardized tools and metrics to produce systems with predictable outcomes

- Mechanical and electrical engineers have big catalogs of standard parts they recycle into their creations vs. "reinventing the wheel"

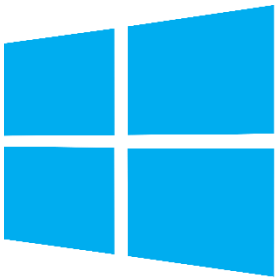
Software Engineering vs. [other] Engineering

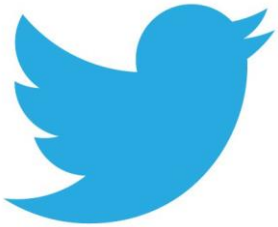
- Other branches of engineering use standardized tools and metrics to produce systems with predictable outcomes

- Mechanical and electrical engineers have big catalogs of standard parts they recycle into their creations vs. "reinventing the wheel"
  - Trust / compatibility of existing software components?

Software Engineering vs. [other] Engineering

"Local" implications





"Global" implications

Properties of Software*

*Brooks, Fred P. (1986). "No Silver Bullet — Essence and Accident in Software Engineering". Proceedings of the IFIP Tenth World Computing Conference: 1069–1076

Properties of Software – Accidental

- Accidental properties change from time to time
- Examples:
  - Programming language
  - Hardware speed, memory size
  - Architecture of the program
    - functional
    - object oriented
- Solutions:
  - High-level programming languages
  - Time-sharing
  - Unified programming environments

Properties of Software – Essential

- Intrinsic to software – determine its nature
- These do not change!

- Complexity
- Conformity/Interoperability
- Changeability
- Invisibility
  - not tangible
  - cannot use senses

Can we mitigate the essential problems of software?

- Ada and other high-level language advances
- Graphical programming
- Object-oriented programming
- Artificial intelligence
- Program verification
- Expert/Recommender systems
- Environments and tools
- "Automatic" programming

New Trends in Software Engineering

- Software evolution
- Agile processes
- Product lines
- Service oriented software
- Software visualization
- Improving OO
- Generative programming
- Empirical software engineering

What to Expect in CS471?

What to Expect in CS471: "Question all the Answers"

What to Expect in CS471: "Question all the Answers"

- Software Engineering is an active area of research
- Best practices continue to emerge from this research
- We supplement our texts with selected research papers

What to Expect in CS471: "Question all the Answers"

- You won't leave CS471 with all the answers
- You will leave thinking critically about the answers!
- You will leave on the trail of continuous education!

CS471: Related Courses

- CS472 provides a deep dive into software design
- CS474 provides a deep dive into software quality
- CS481 (Fall'18) provides a deep dive into a real world software project
- Continuing education following graduation!

HW1 Background Information

Issue Tracking Systems

Bugzilla

GitHub

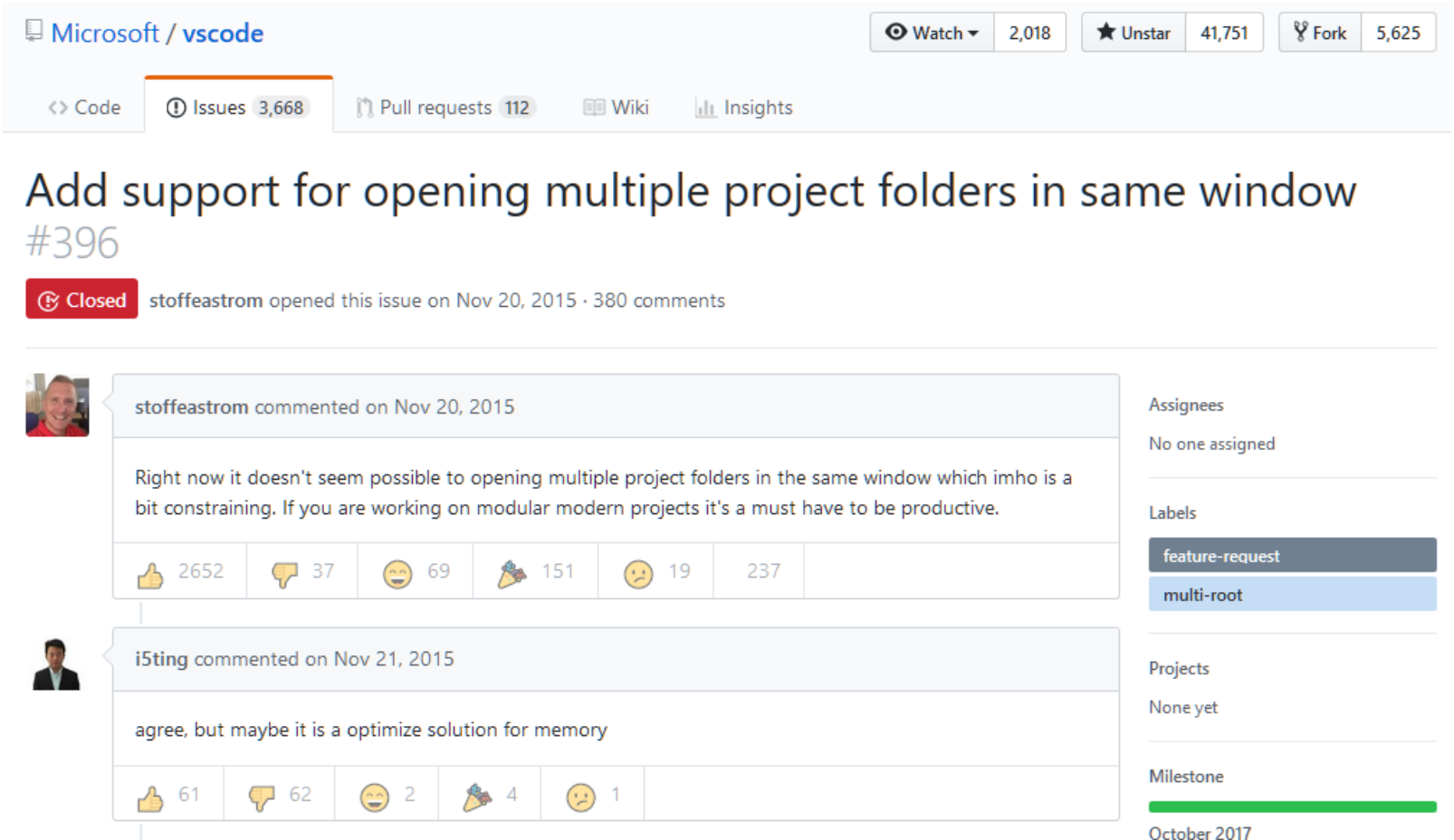
Atlassian JIRA

trac
Integrated SCM & Project Management

- Contain "bug/defect" reports and feature requests

Feature Request

- Describes what new functionality the product should implement
- https://github.com/Microsoft/vscode/issues/396



Bug/Defect Report

- Describes something the product has not correctly implemented

- When writing a bug report, what information should you provide?

- https://developer.mozilla.org/en-US/docs/Mozilla/QA/Bug_writing_guidelines

Defect Report Template for Class Project

Short descriptive title:

Description

Steps to Reproduce:

1. TBD
2.
3.

Actual Results:

TBD

Expected Results:

TBD

Other notes:

TBD

Defect Report Template for Class Project

Short descriptive title:

Description

Steps to Reproduce:

1. TBD
2.
3.

Actual Results:

TBD

Expected Results:

TBD

Other notes:

TBD

Additional information that can be useful:
- environment (OS, platform, version, etc.)
- logs
- memory dumps
- stack traces, etc.

Software Process Models

Software Engineering Process Models

- *Process Model*: Simplified, abstract description of how a software project conducts its activities
  - Specification (Requirements Capture)
  - Software Development (Design and Programming/Implementation)
  - Verification and Validation (Quality)
  - Evolution (Maintenance)

Software Engineering Process Models

- *Process Model*: Simplified, abstract description of how a software project conducts its activities
  - Specification (Requirements Capture)
  - Software Development (Design and Programming/Implementation)
  - Verification and Validation (Quality)
  - Evolution (Maintenance)

- We will mention two models and focus on the second
  - Waterfall
  - Incremental Development (agile)

Waterfall vs. Agile

- Waterfall Model (1970)
  - Plan and make decisions as soon-as-possible
  - Results in long-range plans
  - Original process model

- Agile (Incremental Development) Model (’90s)
  - Plan and make decisions as late-as-possible
  - Results in short-term planning horizons
  - Most popular current model

Caution regarding Process Models

- Both Waterfall and Incremental Development have evolved many adaptations. In CS471, we'll use:
  - Waterfall process as defined in *Software Engineering 10th Edition*
  - Incremental Development as defined in *The Elements of Scrum*

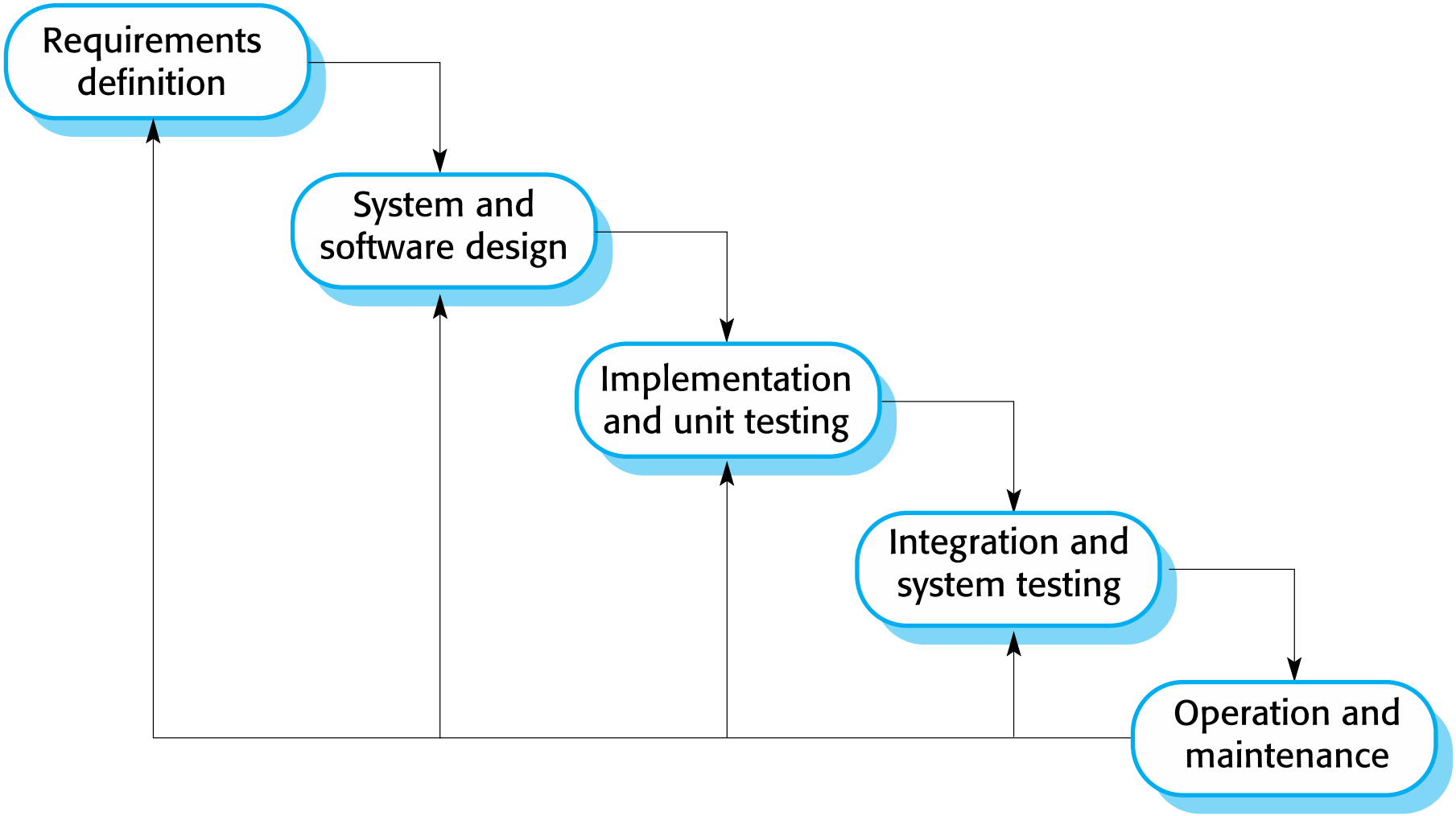
- Your mileage may vary!

Adaptations of Waterfall and Incremental Development (not covered in 471)

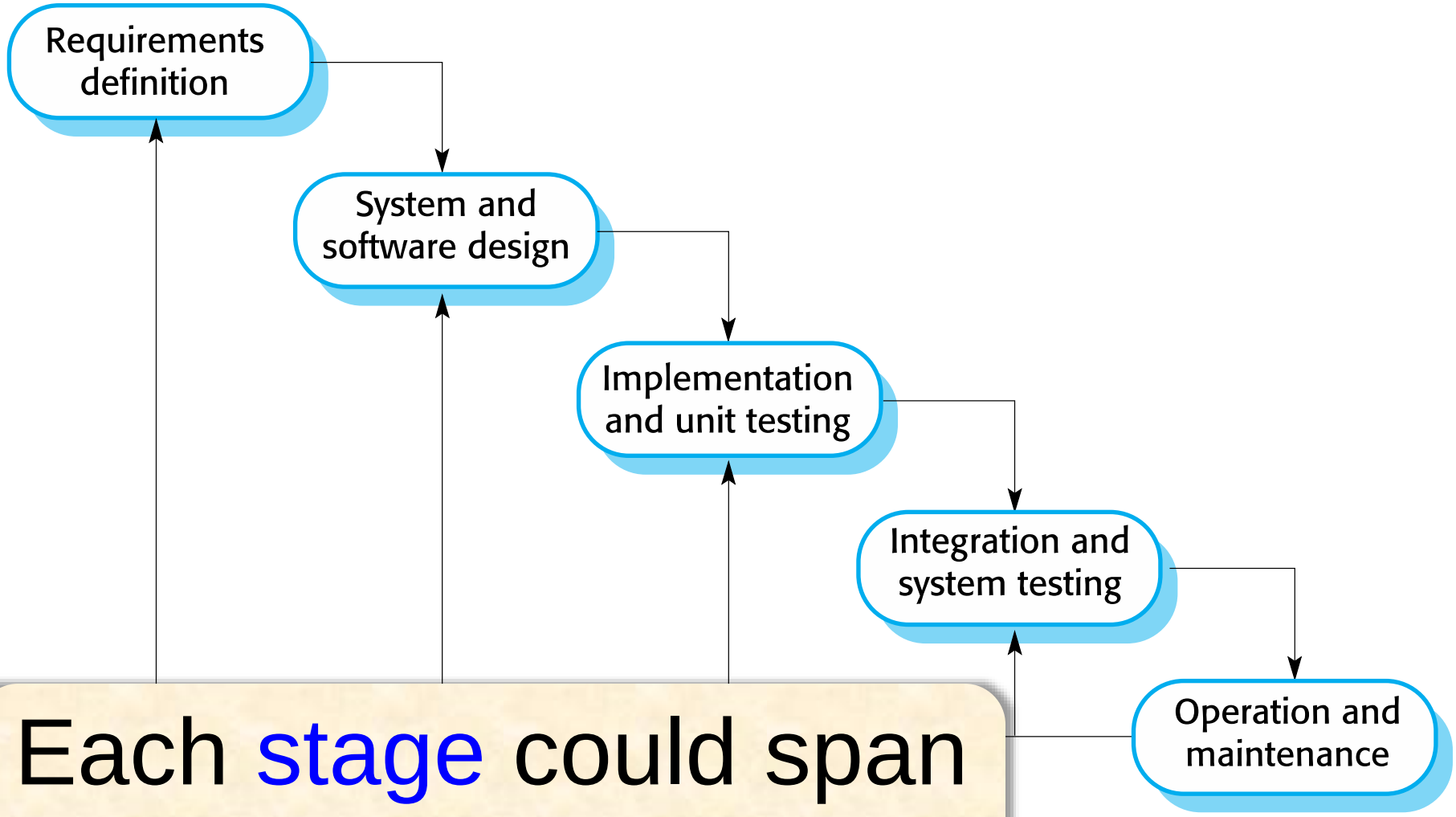
- Prototyping Model
- Rapid Application Development
- Evolutionary Process Models
- Spiral Model
- Component Assembly Model
- Concurrent Development Model
- Formal Methods Model
- Unified Process

The Waterfall Process Model

A Waterfall Process Model [Sommerville]

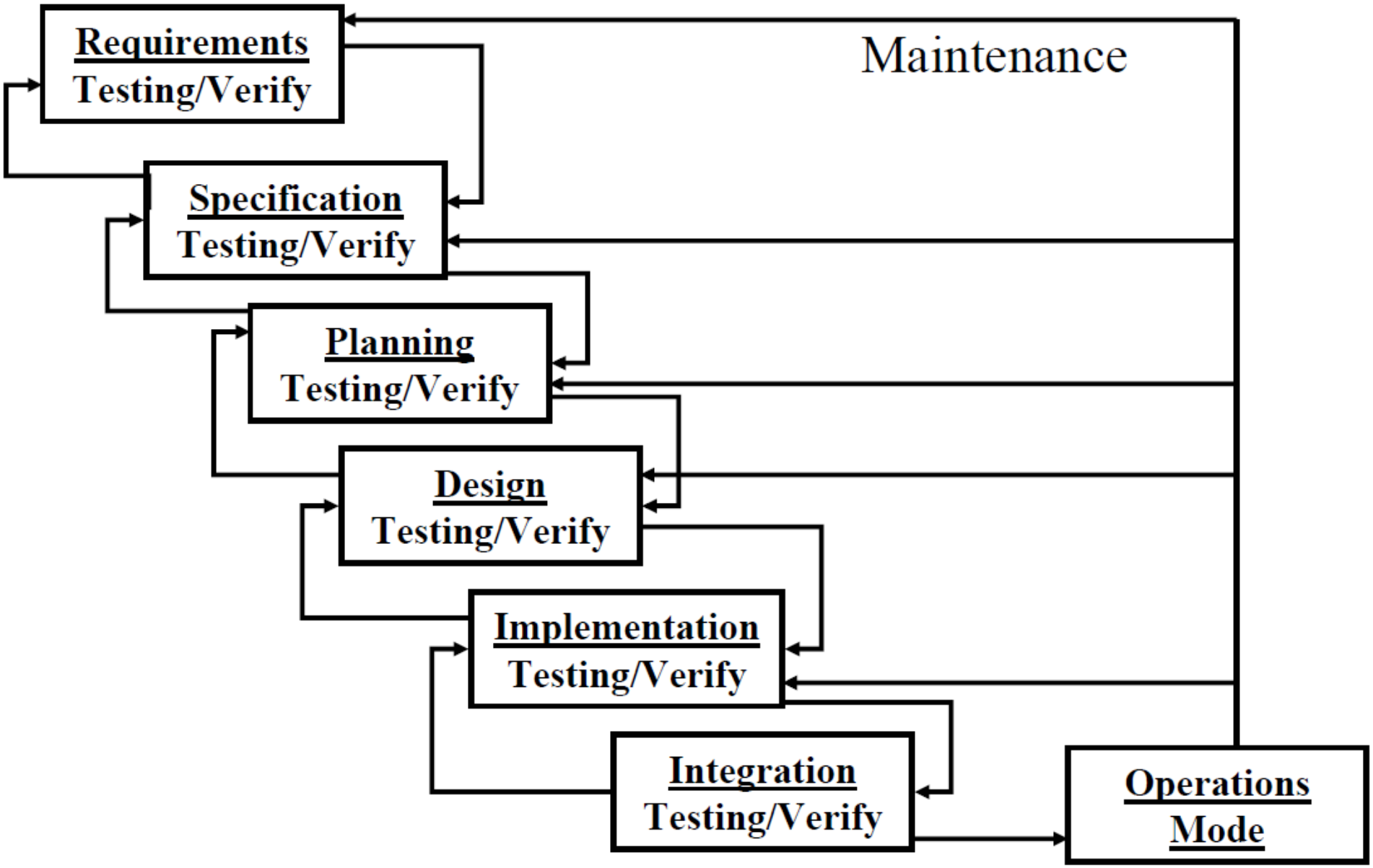


A Waterfall Process Model [Sommerville]



Each stage could span for months or years!

Waterfall Variation



Software Development Activities

- Note: The exact terminology varies somewhat

- We'll try to follow those used in Sommerville

Requirements Capture

- The question that should be asked is …

Requirements Capture

- What does the customer need?
  - Features
  - Usability
  - Reliability/Quality
  - Performance
- What shall we build to fulfill those needs (sometimes called a *specification*)?
- Usually results in a requirements document/list
- Question: How can you determine if the requirements are correct?

Design

- The question that should be asked is …

Design

- How will the software work?
  - Software Architecture (e.g. client/server, layered, etc.)
  - Software design
  - Database design
  - Interface design
  - Reusable (e.g. open-source) component selection
    - Licensing issues

Implementation

- Programming and debugging
- Traditionally an individual activity with no standard process
- Agile challenges that tradition

Testing

- *testing* can be considered as a legacy term

- We will often use the term *defect removal* because modern teams use a variety of defect removal methods beyond testing alone:
  - TBD…

Testing

- *testing* can be considered as a legacy term
- We will often use the term *defect removal* because modern teams use a variety of defect removal methods beyond testing alone:
  - Pair Programming
  - Test-Driven Development
  - Unit-Level Testing
  - Static Analysis
  - Code Reviews
  - Integration and Regression Testing
  - System-Level Testing
- We'll cover these later in the semester

More About Testing

- Defects are vastly cheaper to remove …

More About Testing

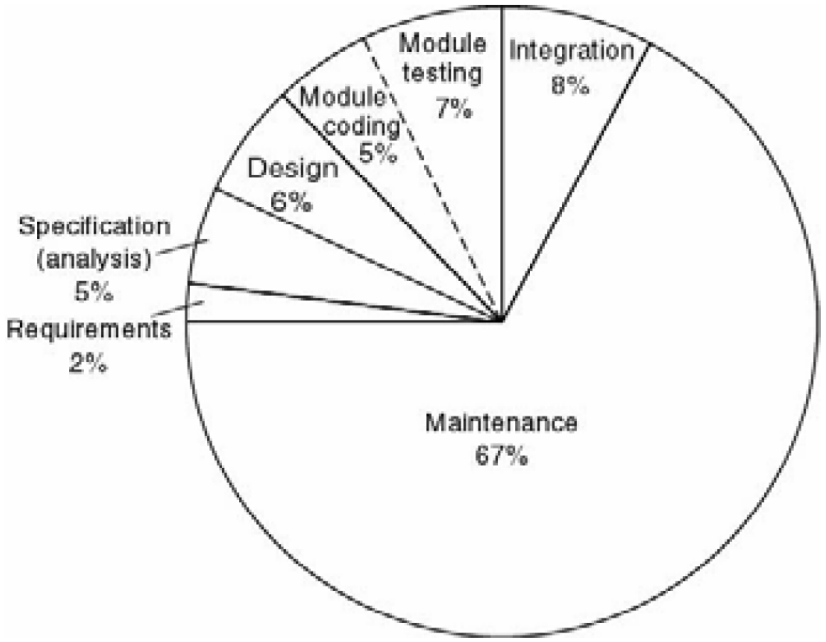
- Defects are vastly cheaper to remove early in the project

Approximate Relative Cost of Each Phase

Approximate Relative Cost of Each Phase

- 1976–1981 data

- Maintenance constitutes 67% of total cost

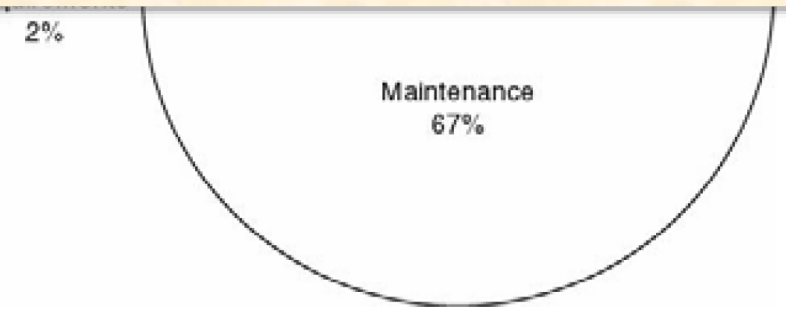


Approximate Relative Cost of Each Phase

- 1976–1981 data

Up to **90%** of software cost is spent on **maintenance**

[Erlikh'00]



Empirical data based on the waterfall model

- Survey by Lientz and Swanson: maintenance activities divided into four classes:
  - Adaptive – changes in the software environment (about 20% of all changes)
  - Perfective – new user requirements (20%)
  - Corrective – fixing errors (20%)
  - Preventive – prevent problems in the future.

Empirical data based on the waterfall model

- 60 to 70% of faults are specification and design faults

- Data of Kelly, Sherif, and Hops [1992]
  - 1.9 faults per page of specification
  - 0.9 faults per page of design
  - 0.3 faults per page of code

Waterfall Characteristics

Waterfall Characteristics

- Activities performed in sequential *stages*
- Gated — Complete current state before beginning the next
- Note heavy up-front planning — *Big Design Up-Front* (BDUF)
- Big Bang Integration

Waterfall Applications

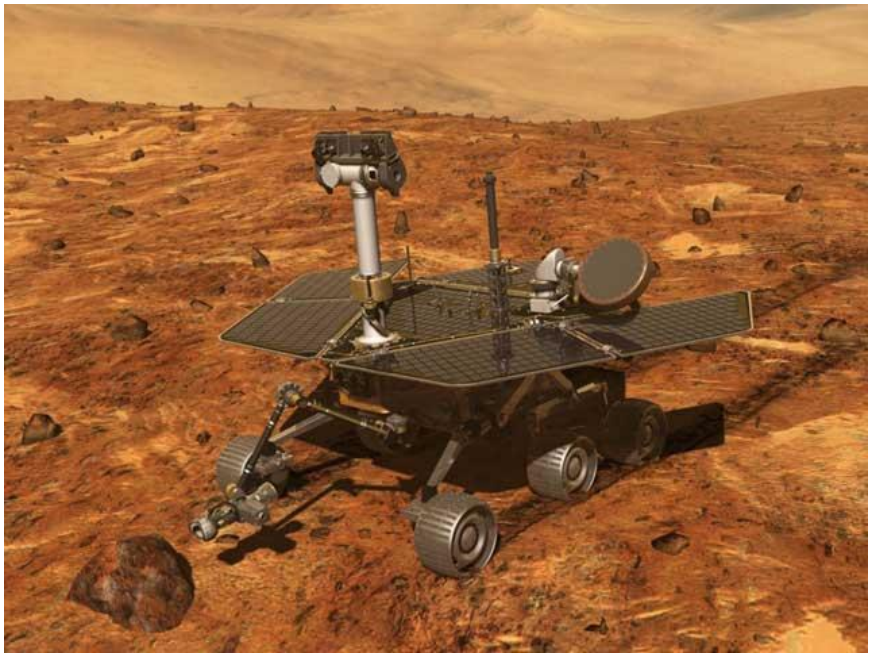
- Works best with stable requirements and technologies
- Not a bad choice for routine IT-like projects
- Arguably useful in any project benefitting from up-front plans

Waterfall Applications

- Arguably the best choice for contractual development
  - Suited executives and their attorneys gather in a conference room
  - Contracts are signed, specifying
    - what will be built,
    - when it will be completed,
    - how much it will cost, and
    - penalties for changes

Waterfall Applications

- Arguably the best choice for contractual development
  - Suited executives and their attorneys gather in a conference room
  - Contracts are signed, specifying
    - what will be built,
    - when it will be completed,
    - how much it will cost, and
    - penalties for changes
  - Widely used in
    - government,
    - aerospace and some
    - enterprise IT



Waterfall Disadvantages and Practical Issues

- Changes waste the painfully created up-front planning
- Inflexible partitioning of development into gated stages
  - may idle resources (e.g., next stage cannot start before current one)
- And, if not used, attempts to achieve the benefits of an agile process without the activities required to be agile

Waterfall Disadvantages and Practical Issues (contd.)

- Big-Bang Integration, if actually used, leads to chaos (Imagine… if we each wrote one *War and Peace* chapter independently, then merged, and sent them to our publisher!)

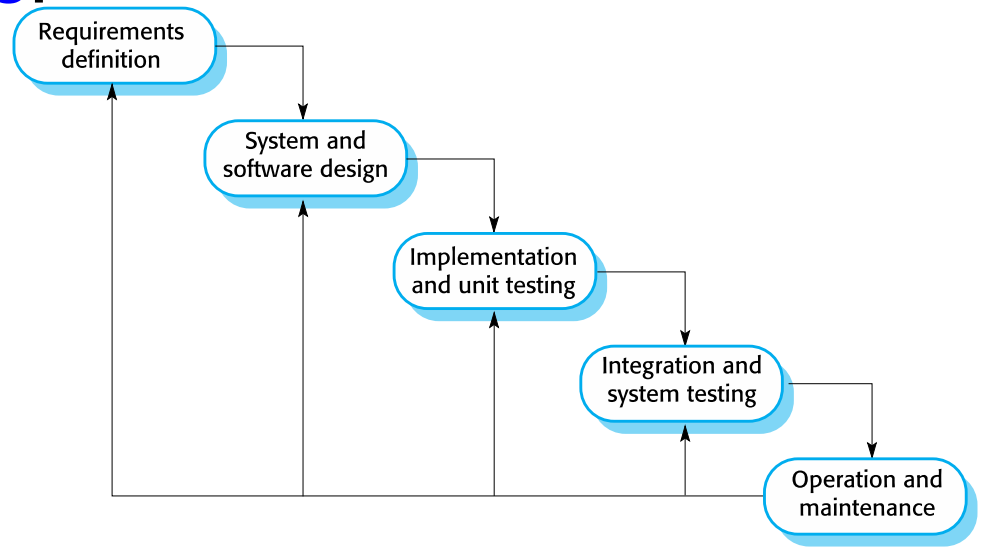
- Invisible problems (e.g., we built the wrong product) may lead to complete failure

Waterfall still exists…

- still a standard
- software engineering textbooks still based on it
- many managers still adhere to it
- rarely followed by the programmers

Waterfall Summary

- Strongly emphasizes up-front planning
- Some phases produce only documents (e.g., requirements)
- Typical sequential phases:
  - Requirements
  - Design
  - Implementation
  - Testing
  - Maintenance



Waterfall Summary

- Each phase built on the planning of a previous phase
- May not produce any code until everything is planned in detail
- Builds the product with "Big bang" integration of code modules

How Would a Team Write *War and Peace*?

How Would a Team Write *War and Peace*?

- English translations contain ≈ 560,000 words
- Tolstoy had 10 years to write *War and Peace* solo

How Would a Team Write *War and Peace*?

- English translations contain ≈ 560,000 words
- Tolstoy had 10 years to write *War and Peace* solo

- 13 Software Engineers developed
  - 115 KLOC (≈ 575,000 "words") in 14 months
  - maybe… perhaps… 575,000 "words" ≈ *War and Peace*

How Would a Team Write *War and Peace*?

- English translations contain ≈ 560,000 words
- Tolstoy had 10 years to write *War and Peace* solo

- 13 Software Engineers developed
  - 115 KLOC (≈ 575,000 "words") in 14 months
  - maybe… perhaps… 575,000 "words" ≈ *War and Peace*

- Imagine writing *War and Peace* using 13 authors in 14 months!

- That's routine for a software engineering team!!!

How Would a Team Write *War and Peace*?

- 13 Software Engineers developed 115 KLOC in 14 months, maybe… perhaps… 575,000 "words" ≈ *War and Peace*

- How many LOC will an engineer write on average per workday? (Solution: whiteboard only)

Group Exercise: Write *War and Peace* in One Year

- Form teams of ~8 people each
- It's 1865 and your publisher has good news and bad news: while agreeing to publish your novel, *War and Peace*, they need the first draft in one year rather than ten

- No problem… you'll simply use as many authors as you need and will be done in a mere fraction of the time!
- Decide how your team will divide the work, coordinate the storyline, and deliver a proofread draft on-time
- Also… your publisher wants French and English drafts in addition to Russian

Another Challenge: Staffing Turmoil

- Your project needs to introduce more competitive features and you're hiring additional developers
- Your competitors' products are more reliable
- You're hiring additional developers to find/fix defects
- Your company is growing and your developers have accepted promotions to new jobs, and you're hiring new developers to replace them
- How will you introduce your new developers to your project?

Another Challenge: Dispersed Teams

- What if management staffed the *War and Peace* project at multiple locations:
  - Different floors of the same building
  - Different building of the same campus
  - Different countries/continents

- How will that complicate development?

Agility: An Emerging Motivation

- Agility refers to our response to changing project conditions
  - Customers don't know what they need
  - The customers' needs change
  - Competitive pressure forces unexpected change
  - Development is more difficult than expected
  - Technologies change
  - Personnel changes
  - Disaster: Equipment failure, weather, fire…
- Business managers can leverage agility to create value ($$$)

Teamwork Arises from Competitive Pressure

- Most software is large: tens, hundreds, even millions of lines of code (LOC)

- 1 KLOC = … LOC
- 1 MLOC = … KLOC = … LOC

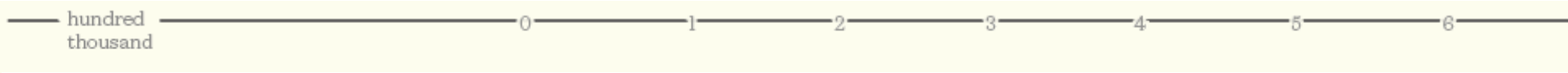
Teamwork Arises from Competitive Pressure

- Most software is large: tens, hundreds, even millions of lines of code (LOC)

- 1 KLOC = 1,000 LOC
- 1 MLOC = 1,000 KLOC = 1,000,000 LOC

Codebases

Hundred thousand
= 100 KLOC
= 100,000LOC



http://www.informationisbeautiful.net/visualizations/million-lines-of-code/

Codebases

Hundred thousand
= 100 KLOC
= 100,000LOC

1 * 100KLOC = 100,000 LOC

2 * 100KLOC = 200,000 LOC

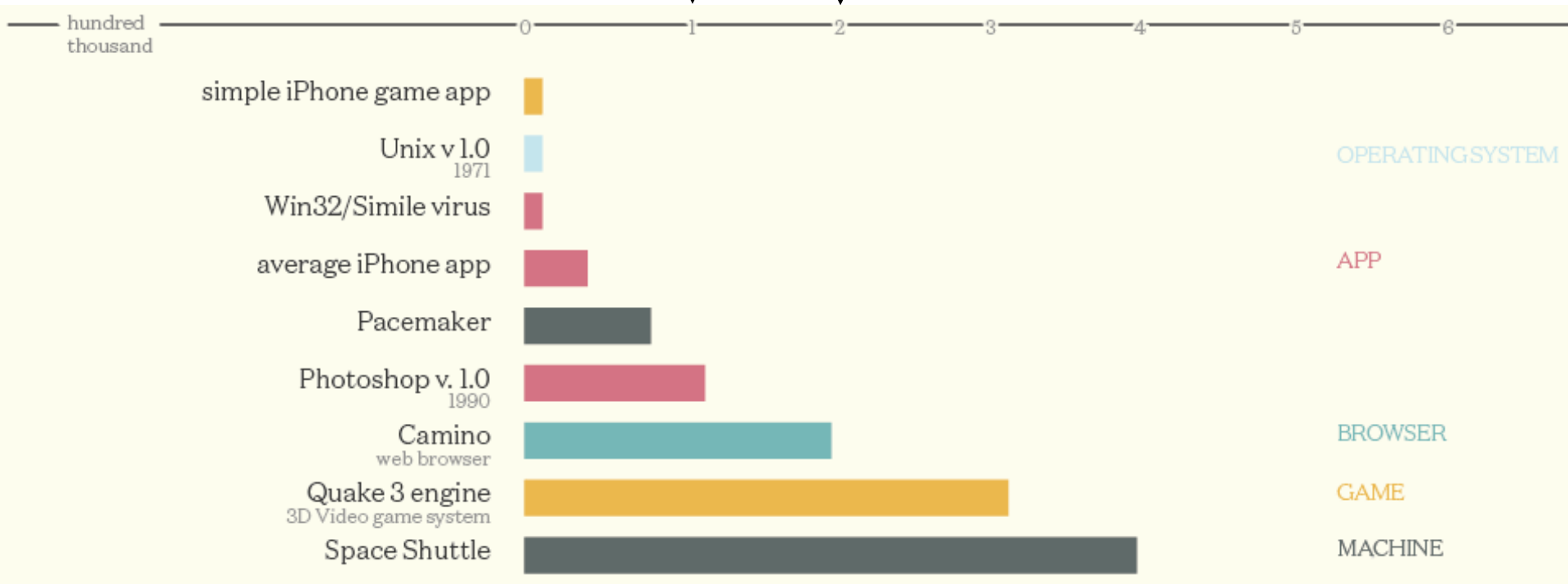
http://www.informationisbeautiful.net/visualizations/million-lines-of-code/

Codebases

Hundred thousand
= 100 KLOC
= 100,000LOC

1 * 100KLOC = 100,000 LOC

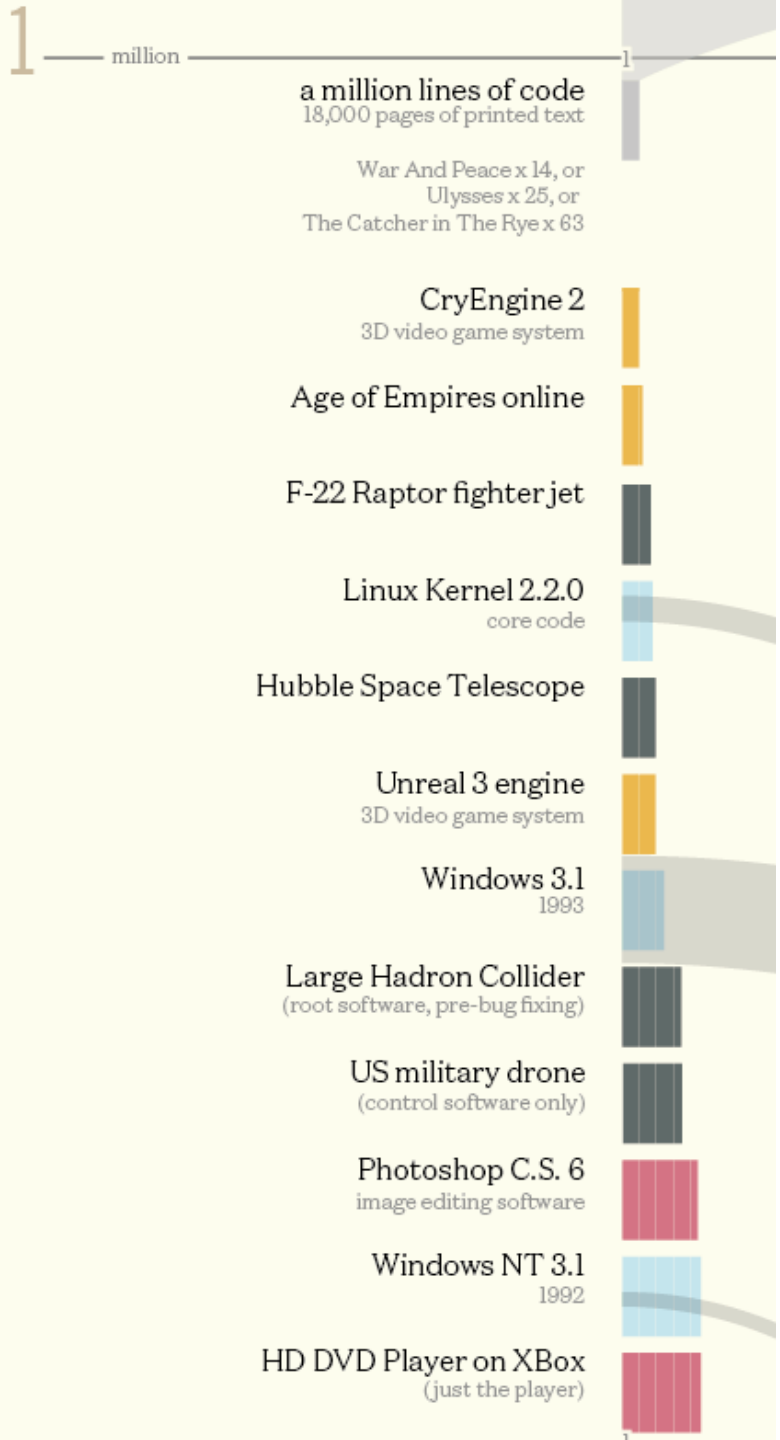
2 * 100KLOC = 200,000 LOC



http://www.informationisbeautiful.net/visualizations/million-lines-of-code/

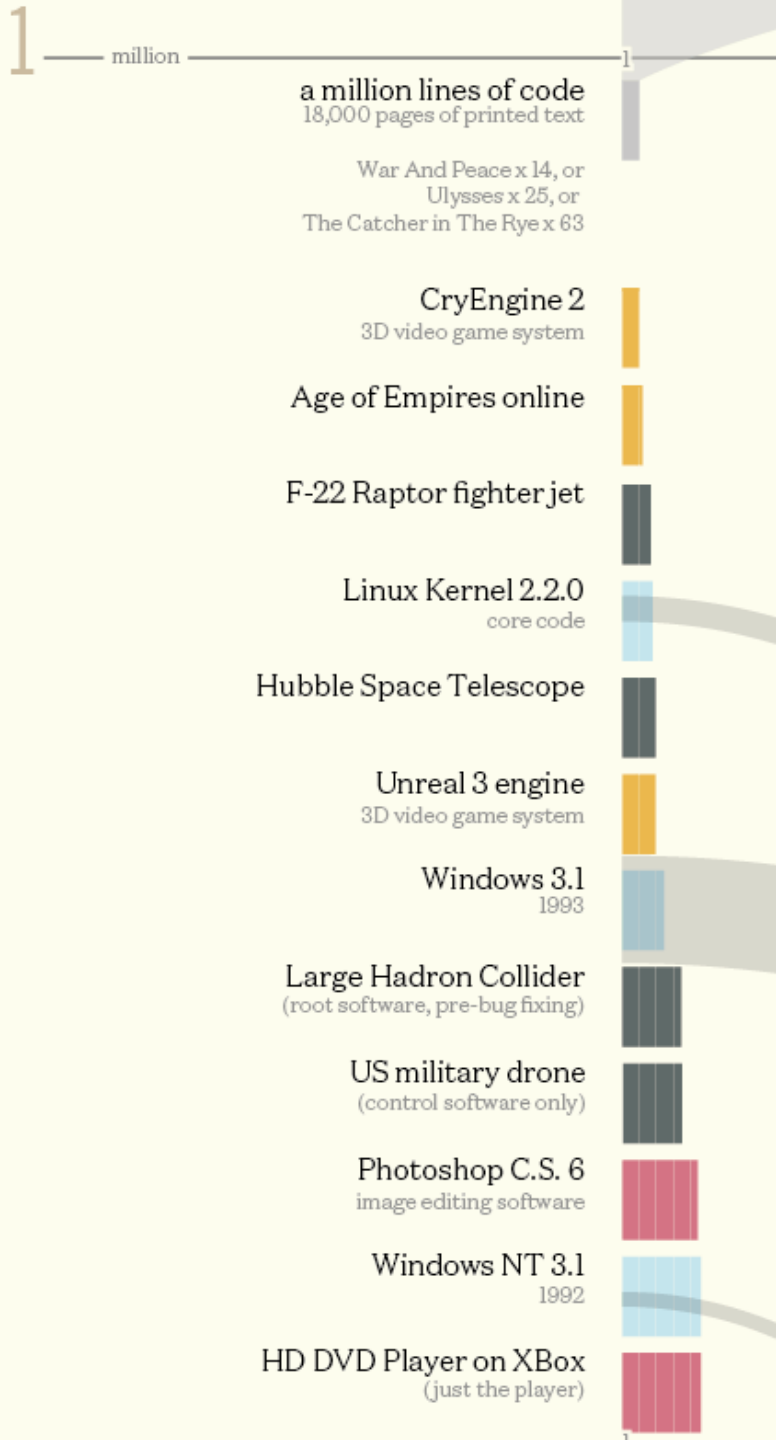
Codebases (MLOC)

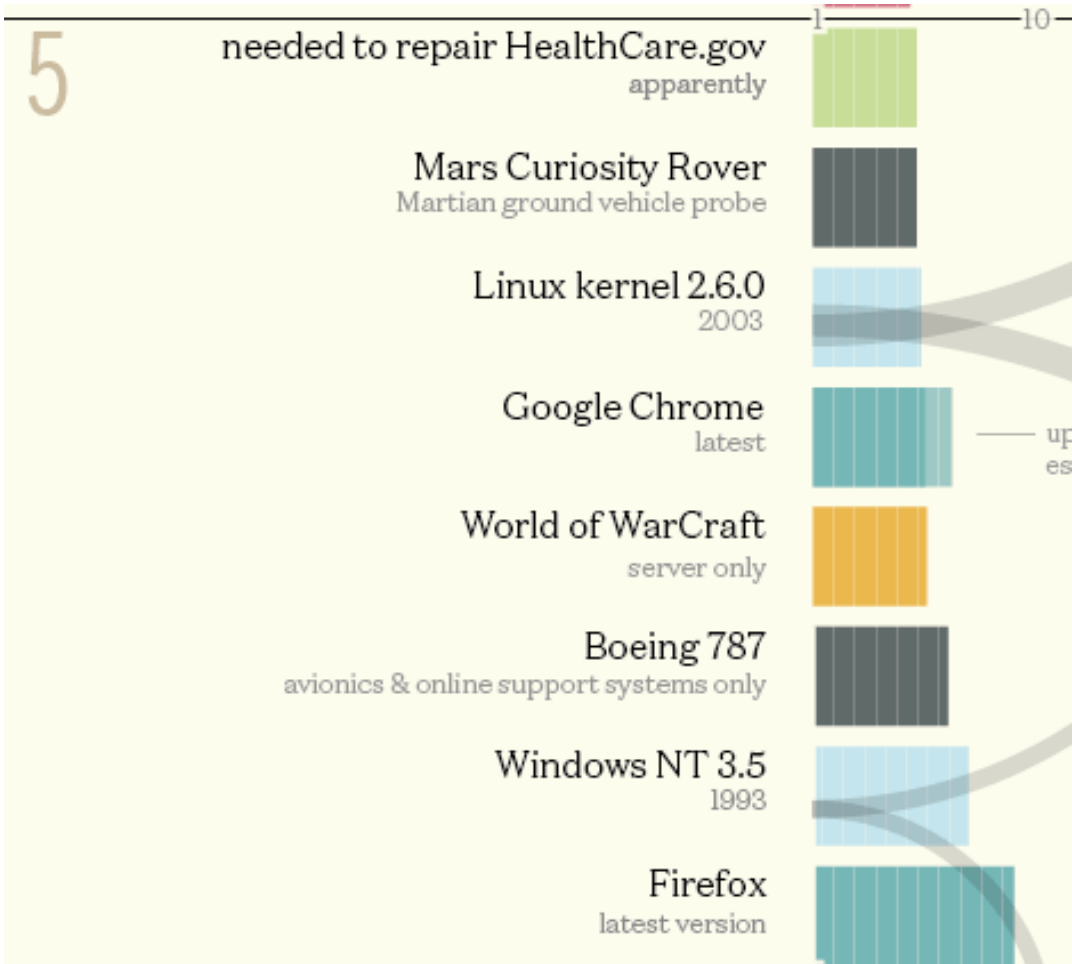
http://www.informationisbeautiful.net/visualizations/million-lines-of-code/



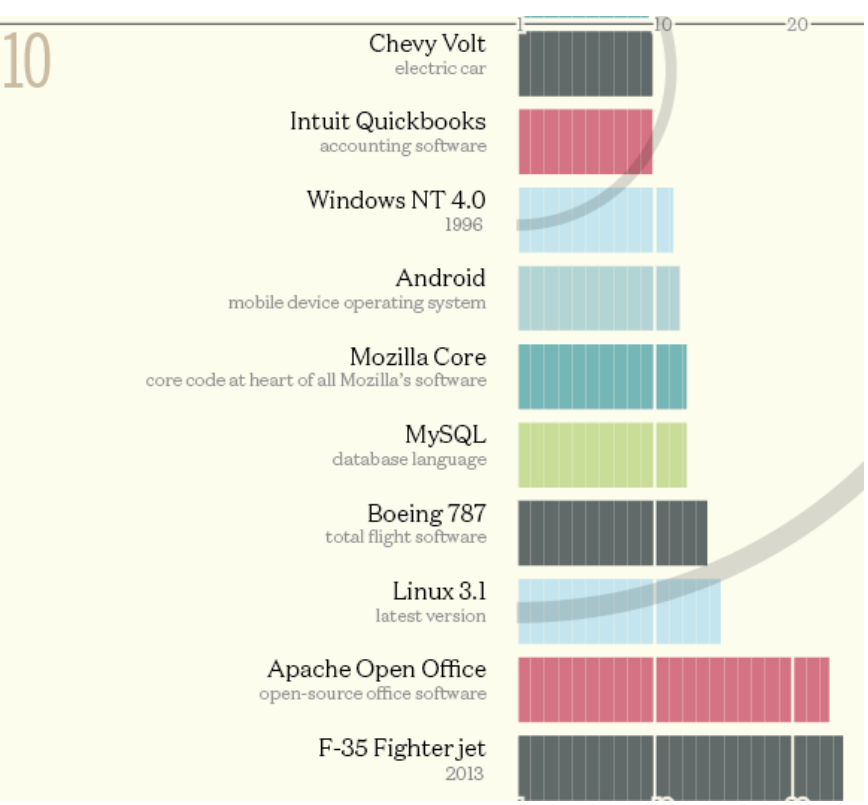
Codebases (MLOC)

http://www.informationisbeautiful.net/visualizations/million-lines-of-code/



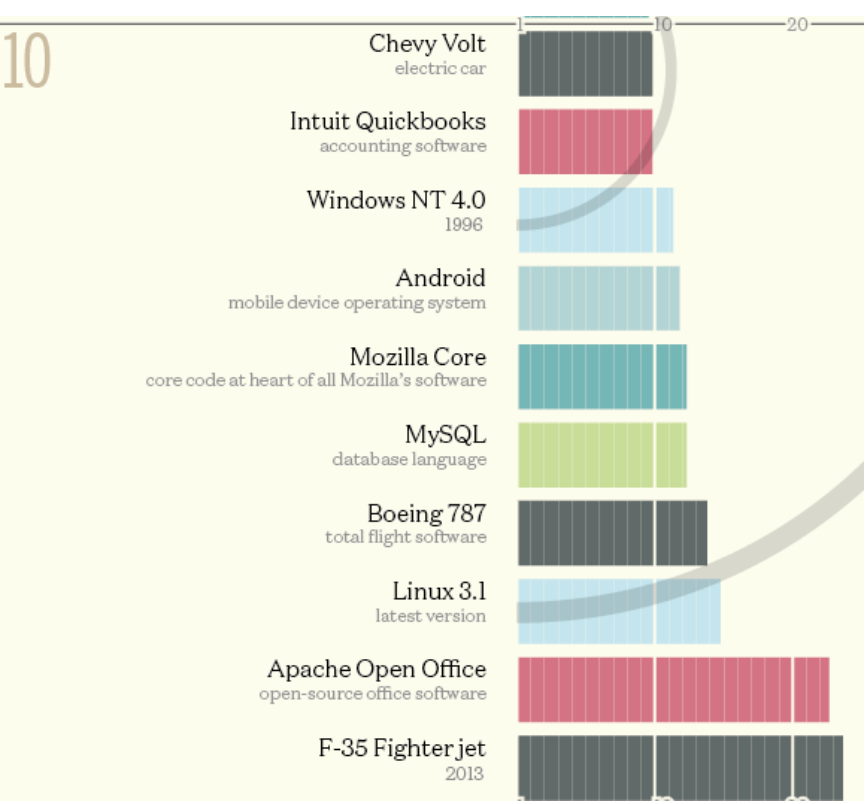


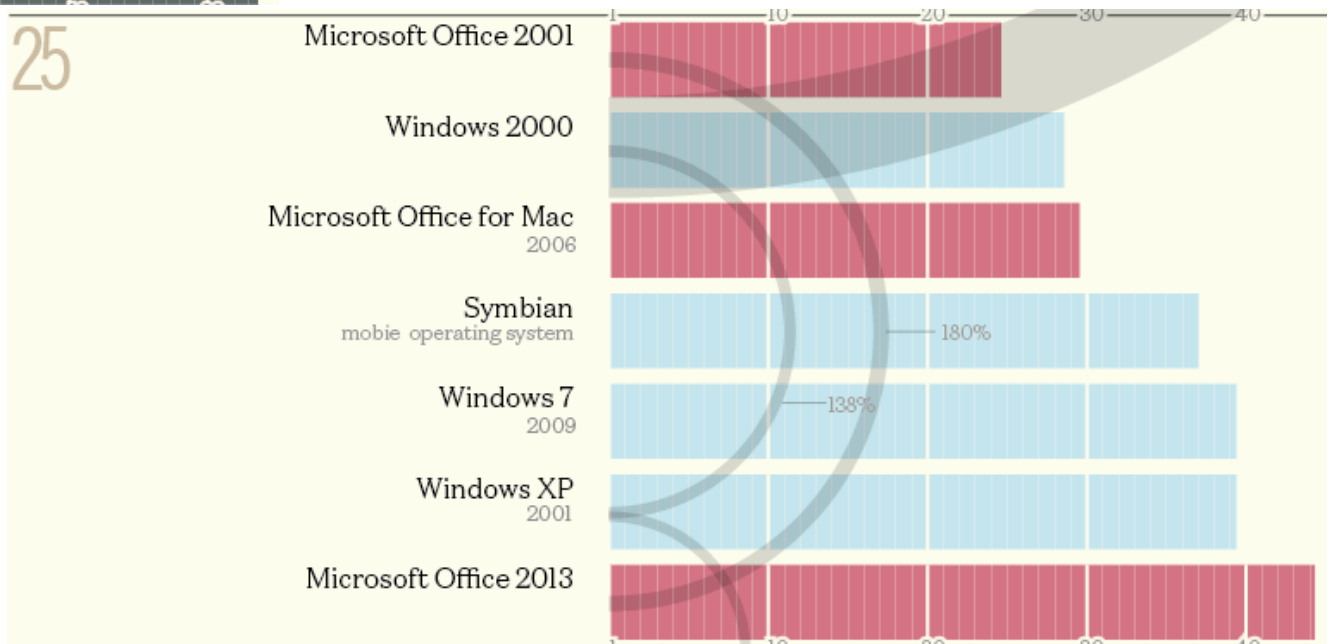
Codebases (MLOC)



http://www.informationisbeautiful.net/visualizations/million-lines-of-code/

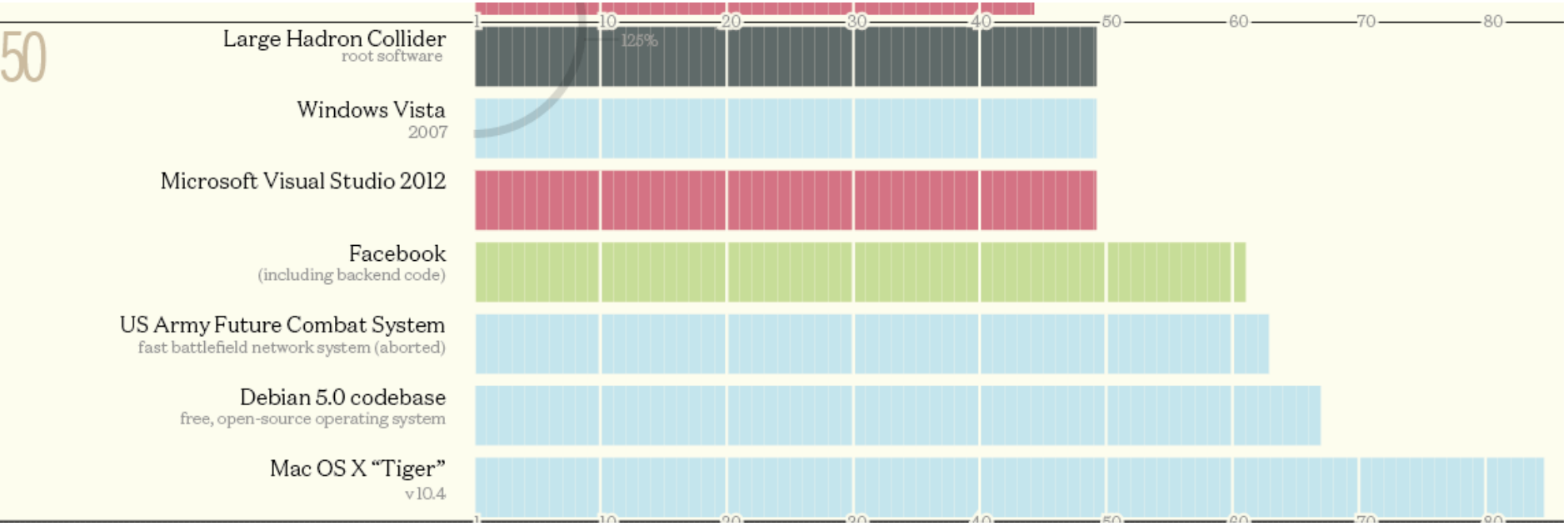
Codebases (MLOC)





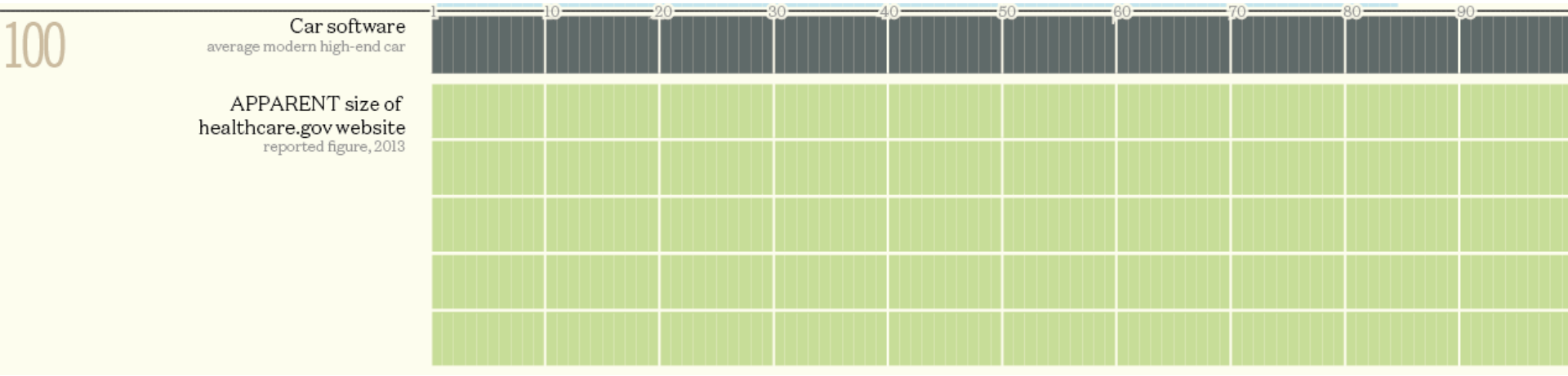
http://www.informationisbeautiful.net/visualizations/million-lines-of-code/

Codebases (MLOC)



http://www.informationisbeautiful.net/visualizations/million-lines-of-code/

Codebases (MLOC)



http://www.informationisbeautiful.net/visualizations/million-lines-of-code/

Google Codebase Size?

http://www.informationisbeautiful.net/visualizations/million-lines-of-code/

Google Codebase Size?

- 2 billion LOC

http://www.informationisbeautiful.net/visualizations/million-lines-of-code/

Teamwork Arises from Competitive Pressure

- Most software is large: tens, hundreds, even millions of lines of code (LOC)

Teamwork Arises from Competitive Pressure

- Most software is large: tens, hundreds, even millions of lines of code (LOC)
- Business can't wait for you to develop all that code alone
- You will work in a team
- Some of you will work in a network of teams, some who you will never meet in person (think… Open Source development)

Incremental Development

Incremental Development

- Product is constructed in incremental releases rather than a single release
  - Releases, but not activities, are usually gated
    - what do we mean by gated?

Incremental Development

- Product is constructed in incremental releases rather than a single release
  - Releases, but not activities, are usually gated (i.e., new changes are not allowed for the current release)

- Each release incorporates a set of activities
  - Requirements
  - Design
  - Implementation
  - Testing, etc.

Incremental Development

- Initial release often delivers a very minimal experience (e.g., User Interface with nothing behind it)

- Subsequent releases deliver increasing functionality

Incremental Development Benefits

- (from the client/customer point of view…)

Incremental Development Benefits

- Agility reduces the cost of responding to change
  - The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model
  - Change is "expected"

- Customers validate incremental releases
  - Customers comment on use or demonstrations of the software and see how much has been implemented
  - Project often identifies and recovers from problems early

Incremental Development Benefits (contd)

- More rapid delivery and deployment of useful software to the customer is possible
  - Customers use and gain value from early releases
  - "First to market" advantage
  - Features can be implemented in prioritized order

Incremental Development Limitations

Incremental Development Limitations

- The software design rots without regular refactoring, making future changes more expensive over what becomes a legacy design
  - Solution: Routinely include "code health" activities and engineering practices

Incremental Development Practice

- Scrum is the most widely used incremental development process
  - We'll spend several weeks on it

Waterfall vs. Incremental/Agile (CHAOS report from Standish Group)

CHAOS report from Standish Group

- between 1994-2004: 66% - 84% of software projects were either:

CHAOS report from Standish Group

- between 1994-2004: 66% - 84% of software projects were either:
  - canceled before completion or completed late,
  - over budget, or
  - with features missing

- Large percentage of failures

2015 CHAOS report from Standish Group

- 50K projects studied

https://www.infoq.com/articles/standish-chaos-2015

2015 CHAOS report from Standish Group

- 50K projects studied

MODERN RESOLUTION FOR ALL PROJECTS

	2011	2012	2013	2014	2015
SUCCESSFUL	29%	27%	31%	28%	29%
CHALLENGED	49%	56%	50%	55%	52%
FAILED	22%	17%	19%	17%	19%

The Modern Resolution (OnTime, OnBudget, with a satisfactory result) of all software projects from FY2011–2015 within the new CHAOS database. Please note that for the rest of this report CHAOS Resolution will refer to the Modern Resolution definition not the Traditional Resolution definition.

https://www.infoq.com/articles/standish-chaos-2015

2015 CHAOS report from Standish Group

CHAOS RESOLUTION BY PROJECT SIZE

	SUCCESSFUL	CHALLENGED	FAILED
Grand	2%	7%	17%
Large	6%	17%	24%
Medium	9%	26%	31%
Moderate	21%	32%	17%
Small	62%	16%	11%
TOTAL	**100%**	**100%**	**100%**

The resolution of all software projects by size from FY2011–2015 within the new CHAOS database.

https://www.infoq.com/articles/standish-chaos-2015

2015 CHAOS report from Standish Group

CHAOS RESOLUTION BY AGILE VERSUS WATERFALL

SIZE	METHOD	SUCCESSFUL	CHALLENGED	FAILED
Small Size Projects	Agile	58%	38%	4%
	Waterfall	44%	45%	11%

The resolution of all software projects from FY2011–2015 within the new CHAOS database, segmented by the agile process and waterfall method. The total number of software projects is over 10,000.

https://www.infoq.com/articles/standish-chaos-2015

2015 CHAOS report from Standish Group

CHAOS RESOLUTION BY AGILE VERSUS WATERFALL

SIZE	METHOD	SUCCESSFUL	CHALLENGED	FAILED
Medium Size Projects	Agile	27%	62%	11%
	Waterfall	7%	68%	25%
Small Size Projects	Agile	58%	38%	4%
	Waterfall	44%	45%	11%

The resolution of all software projects from FY2011–2015 within the new CHAOS database, segmented by the agile process and waterfall method. The total number of software projects is over 10,000.

https://www.infoq.com/articles/standish-chaos-2015

2015 CHAOS report from Standish Group

CHAOS RESOLUTION BY AGILE VERSUS WATERFALL

SIZE	METHOD	SUCCESSFUL	CHALLENGED	FAILED
Large Size Projects	Agile	18%	59%	23%
	Waterfall	3%	55%	42%
Medium Size Projects	Agile	27%	62%	11%
	Waterfall	7%	68%	25%
Small Size Projects	Agile	58%	38%	4%
	Waterfall	44%	45%	11%

The resolution of all software projects from FY2011–2015 within the new CHAOS database, segmented by the agile process and waterfall method. The total number of software projects is over 10,000.

https://www.infoq.com/articles/standish-chaos-2015

2015 CHAOS report from Standish Group

CHAOS RESOLUTION BY AGILE VERSUS WATERFALL

SIZE	METHOD	SUCCESSFUL	CHALLENGED	FAILED
All Size Projects	Agile	39%	52%	9%
	Waterfall	11%	60%	29%
Large Size Projects	Agile	18%	59%	23%
	Waterfall	3%	55%	42%
Medium Size Projects	Agile	27%	62%	11%
	Waterfall	7%	68%	25%
Small Size Projects	Agile	58%	38%	4%
	Waterfall	44%	45%	11%

The resolution of all software projects from FY2011–2015 within the new CHAOS database, segmented by the agile process and waterfall method. The total number of software projects is over 10,000.

https://www.infoq.com/articles/standish-chaos-2015