COSC 340: Software Engineering

Software Processes

Prepared by Michael Jantz

(adapted from slides by Ravi Sethi, University of Arizona)

Software Development

- "Software development … requires choosing the most appropriate tools, methods, and approaches for a given development environment."
  - ACM / IEEE 2013 Curriculum Guidelines

Source: https://xkcd.com

Introduction to Processes

- Dictionary definition of process:
  - A series of actions to produce a result
- Our definition of *software development process*:
  - A systematic method for meeting the goals of a project by:
    - (who): determining the roles and skills needed to do the project
    - (what): defining and organizing development activities
    - (when): setting criteria and deadlines for progressing from one activity to the next
- Note: activities in a software process may be sequential, concurrent, or iterative.

Process Assurances

- The system does what the customer/market wants or what the organization wants to build
- Work assignments are properly apportioned
- The code works; e.g., an individual module correctly implements its interface
- The code works together; e.g., modules work together to produce the desired result or that requirements are met
- Teams at different sites understand the interactions among their work and the work at other sites
- Work assignments can be completed in time for the project to meet its release date(s)

How to Find the Right Process?

CUSTOMER-RELATED

Requirements

Application: critical or discretionary?
Requirements: fluid or fixed?

TEAM-RELATED

Development

Team: experienced?
Co-located or distributed?

Process?

PRODUCT-RELATED

Technology

Architecture: flexible?
Tools: proven or risky?

Constraints

Priority: cost, schedule, or quality?
Regulations or standards?

ORGANIZATION-RELATED

Process Classes

- Plan and document
  - Assumes that well-defined stable requirements, documented in advance, guide all phases of development through maintenance
  - Infrequent, often indirect, interactions among stakeholders
- Iterative and incremental
  - A sequence of iterations, each building on the last
  - An iteration is a distinct sequence of activities based on an evaluation plan, resulting in an executable release (internal or external)
- Agile: quick Iterations, in week(s), not months
  - Early and continuous delivery of small increments of valuable software
  - Welcome changing requirements, even late in development
  - Frequent direct interactions among stakeholders

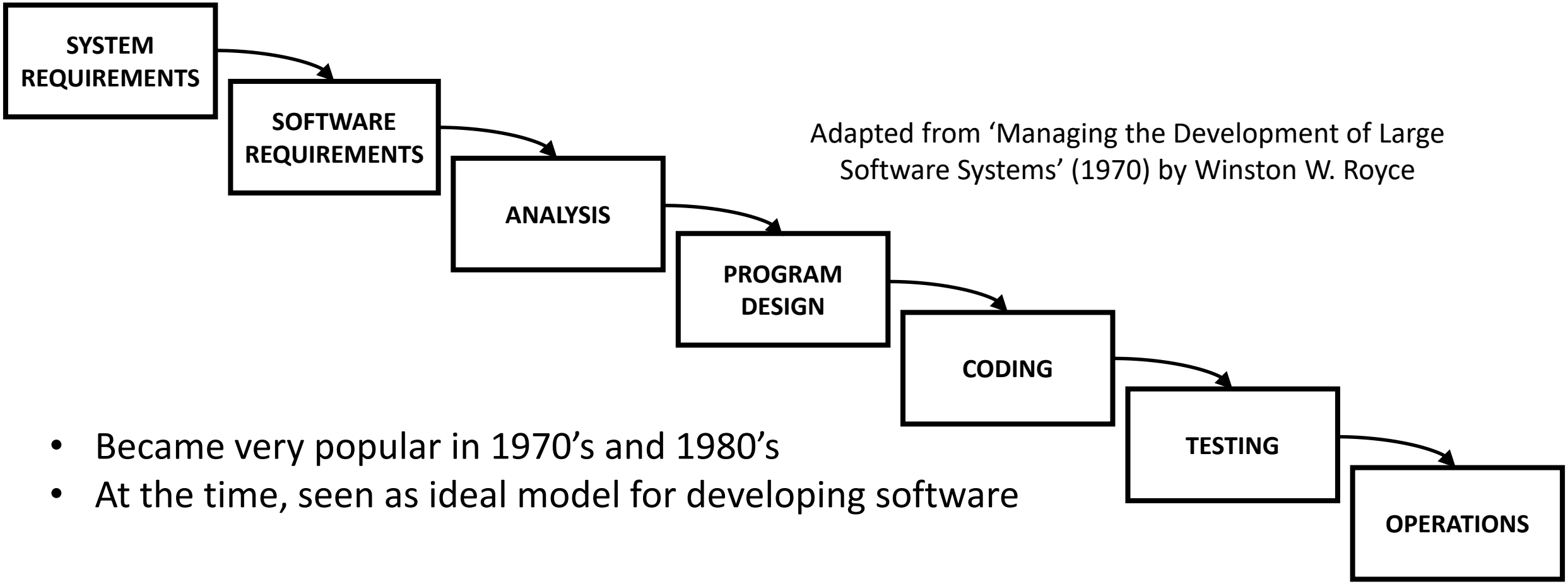
Processes: Plan and Document

- Assume well-defined written specifications guide all phases of development and maintenance
- Dominant model for software in the 1970's and into the 1980's
- Work well when two preconditions are met:
  - Requirements are relatively stable
  - Phases of development can be planned up front

Plan and Document Class

- Definition: divide software development into phases: e.g.,
  - Requirements elicitation
  - Design
  - Coding
  - Testing
- Also called 'Plan and Document'
- Instances we will discuss:
  - Waterfall: proceed sequentially through the phases – very risky
  - V-Processes: separate test planning from text execution
  - Disposable prototype + Waterfall / V-Process: prototype to reduce risk
  - Sequence of Waterfall versions: freeze requirements for each version

Waterfall Process: Sequential Activities



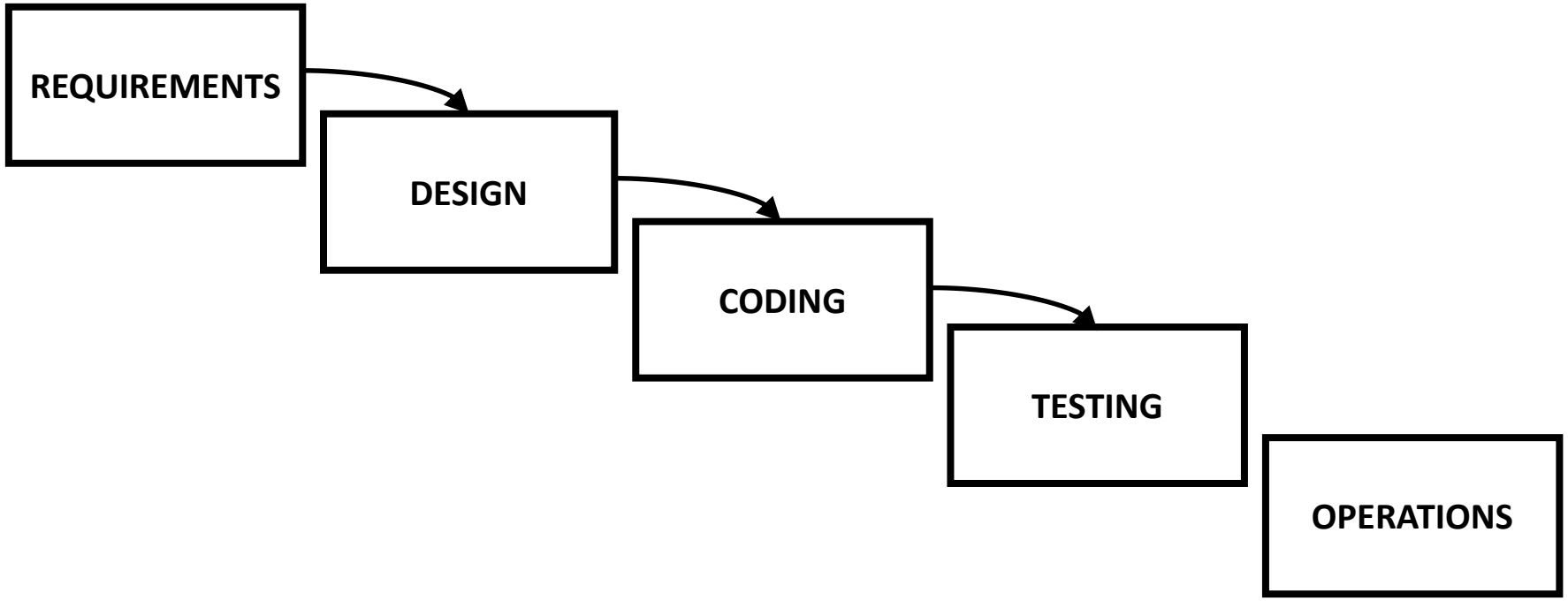
Adapted from 'Managing the Development of Large Software Systems' (1970) by Winston W. Royce

- Became very popular in 1970's and 1980's
- At the time, seen as ideal model for developing software

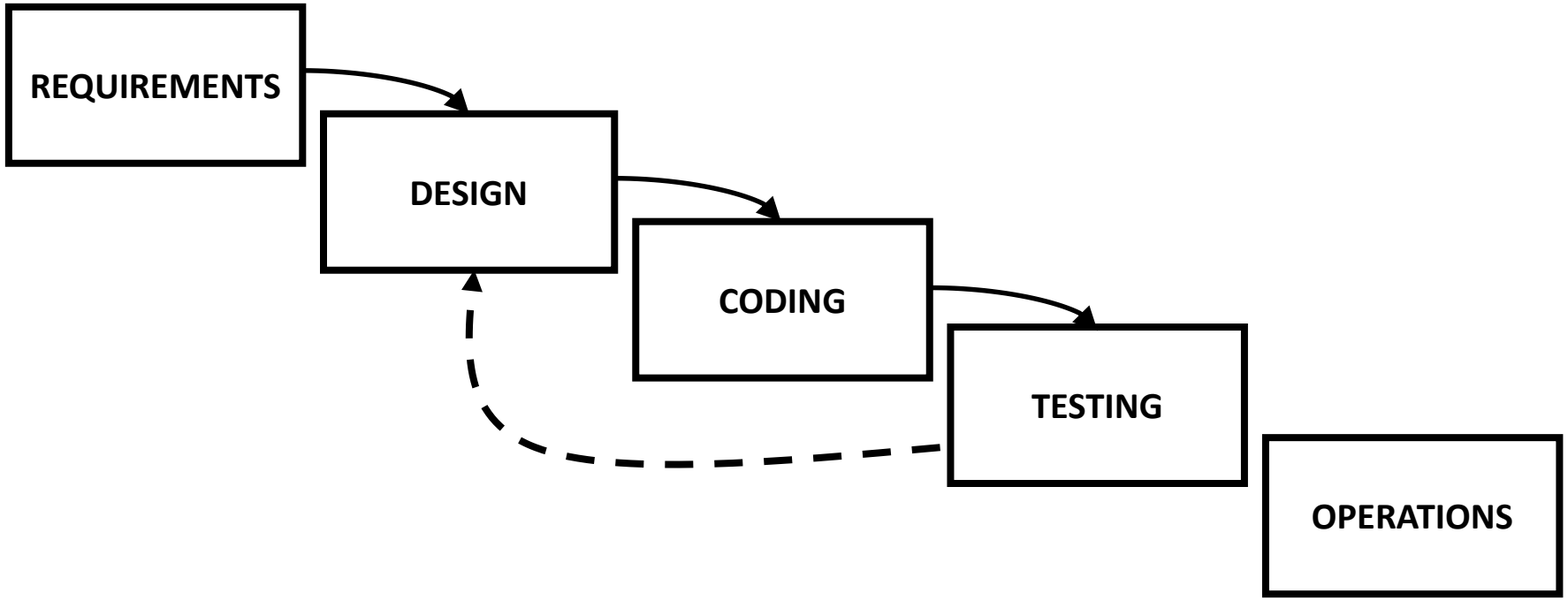
Problems with Pure Waterfall

- No customer involvement after the Requirements phase
  - Requirements often change during software development
- Testing does not occur until the end
  - No feedback during development
  - "We tend to go on for years, with tremendous investments to find that the system which was not well understood to begin with does not work as anticipated" – Robert Graham (MIT), 1968
  - Result: redesign and cost overruns

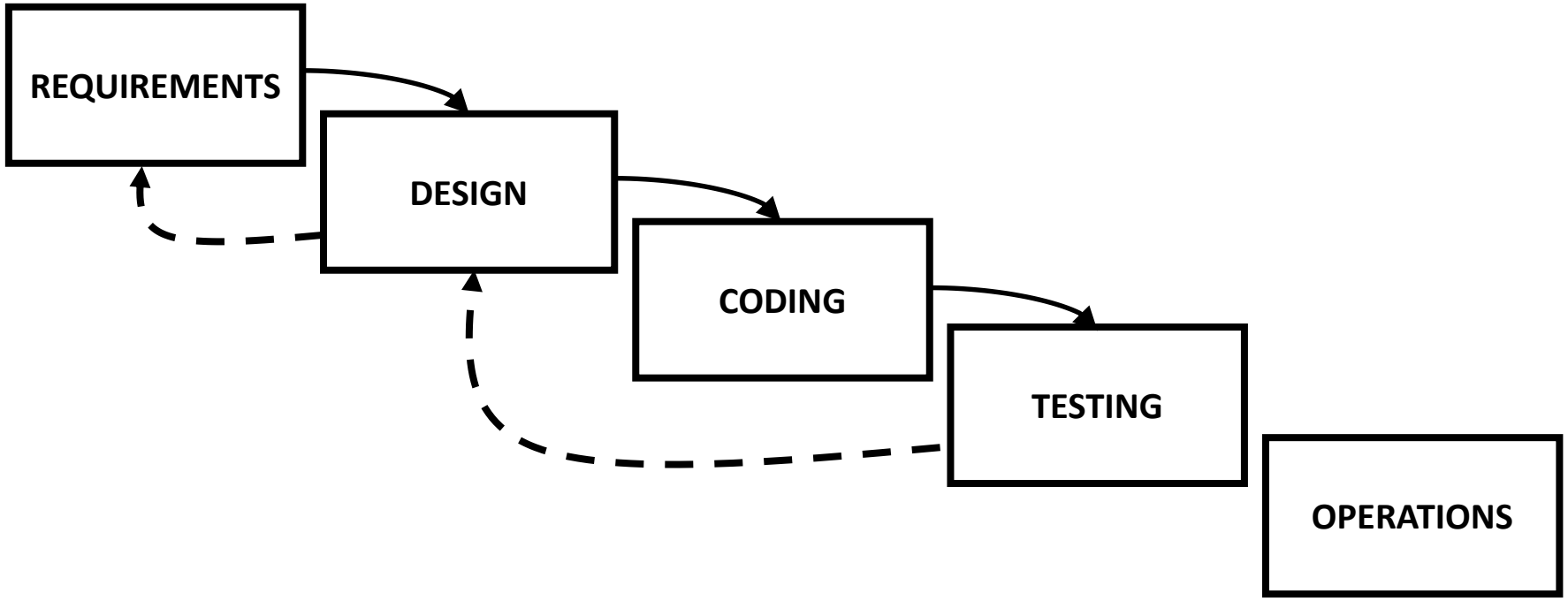
Waterfall Process Requires Backtracking



Waterfall Process Requires Backtracking

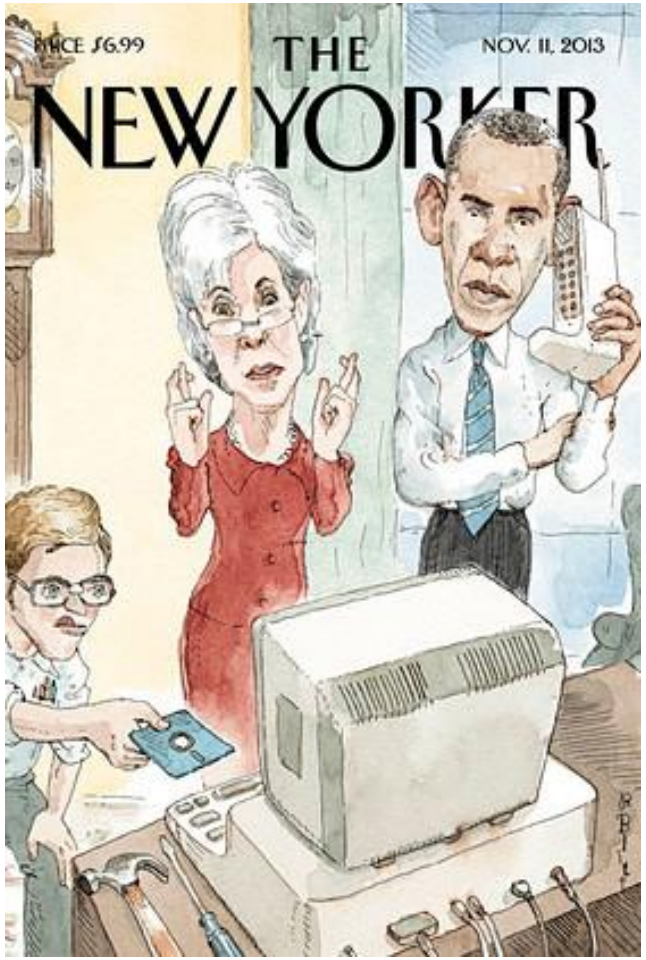


Waterfall Process Requires Backtracking



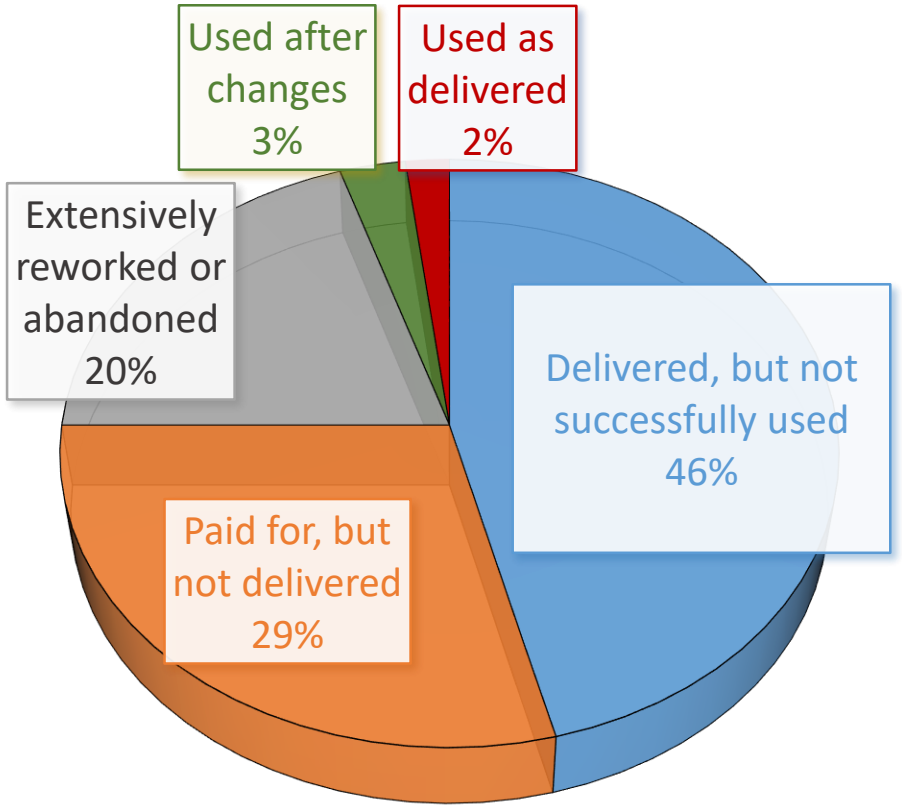
Troubled Launch of HealthCare.gov

- Launch on 10/1/13 marred with serious technical problems
- Contractors admitted that integration testing for the website began two weeks before launch
- Another contractor admitted that full end-to-end system testing did not occur until "the couple of days leading up to the launch"
- The original budget of $93 million had already grown to $292 million by the time of the launch
  - Inspector General report found total cost eventually reached $1.7 billion



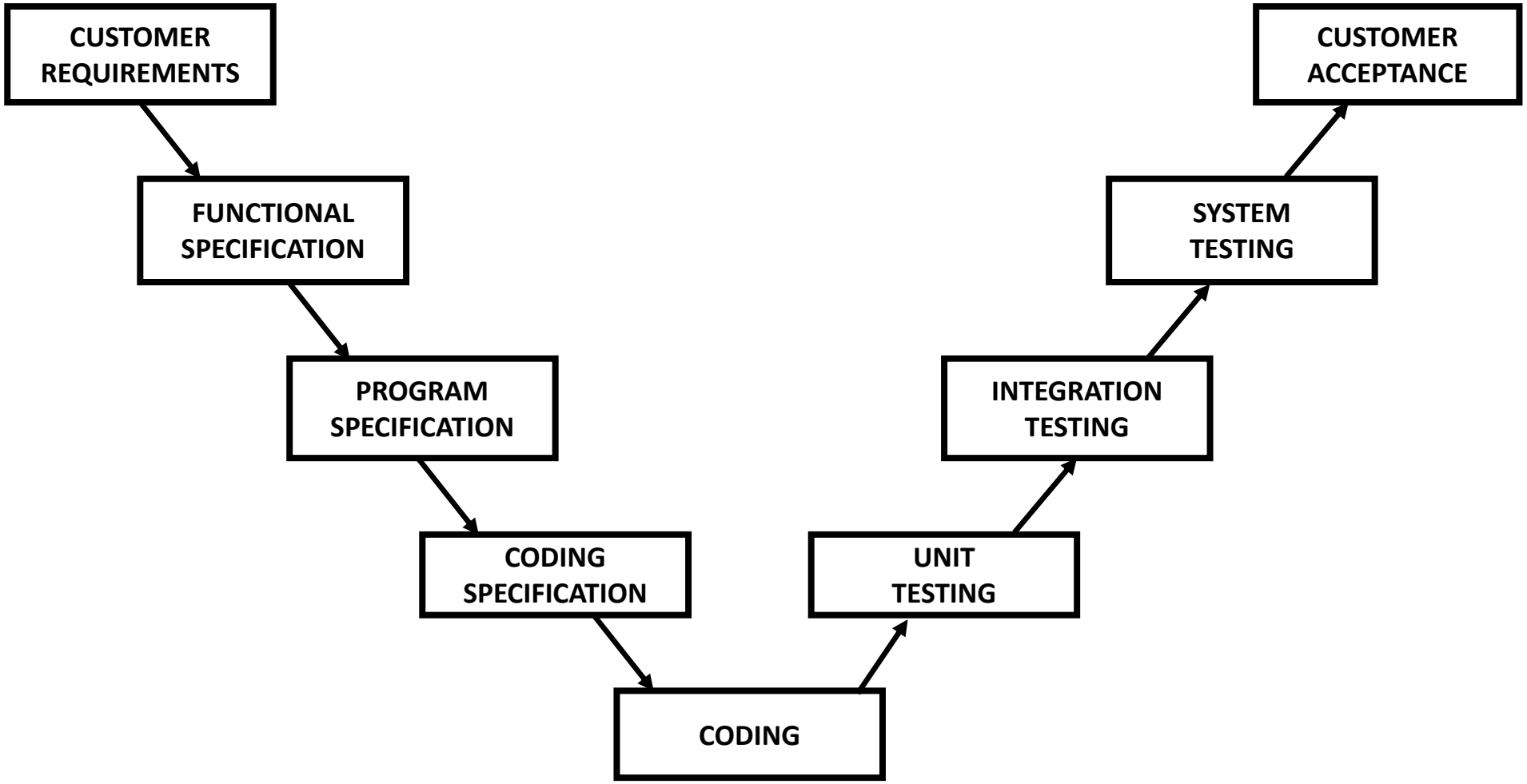
DoD Experience with Software Acquisition

- 1999 review of a sample of projects based on Waterfall (cost $35.7 B)

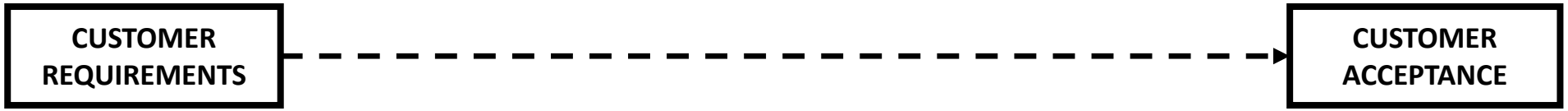


Source: Jarzombek [1999]

V-Process


CUSTOMER REQUIREMENTS
FUNCTIONAL SPECIFICATION
PROGRAM SPECIFICATION
CODING SPECIFICATION
CODING
UNIT TESTING
INTEGRATION TESTING
SYSTEM TESTING
CUSTOMER ACCEPTANCE

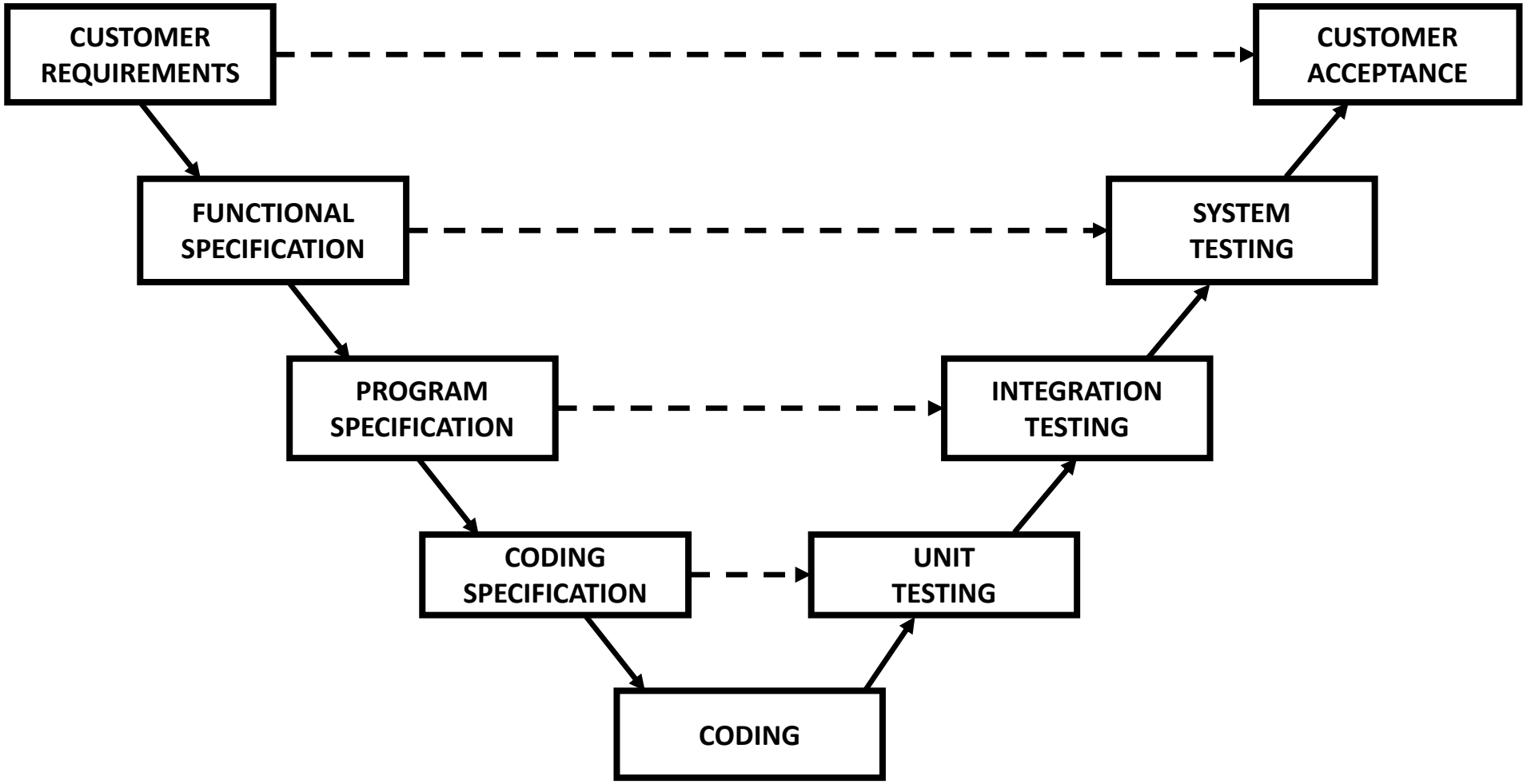
V-Process



Acceptance tests planned here

Acceptance tests run here

V-Process


CUSTOMER REQUIREMENTS
CUSTOMER ACCEPTANCE
FUNCTIONAL SPECIFICATION
SYSTEM TESTING
PROGRAM SPECIFICATION
INTEGRATION TESTING
CODING SPECIFICATION
UNIT TESTING
CODING

SAGE: Air Defense System: 1950s

SAGE: Air Defense System: 1950s

- Semi-Automated Ground Environment (SAGE)
  - Successful system used disposable prototype + V-process
- Ambitious distributed system grew to
  - 24 radar data collection centers and 3 combat centers across the US
  - 100,000 lines of assembly code
- Disposable prototype with just enough functionality
  - To explore tradeoffs between performance, cost, and risk
  - "Twenty people understood in detail the performance of those 35,000 instructions; they knew what each module would do, they understood the interfaces, and they understood the performance requirements."

    Bennington [1956]

Multiple Plan-Driven Releases

- Employ experience from releases 1 though *n* to plan release *n* + 1
- Used to implement AT&T 5ESS telephone switching system

AT&T 5ESS Switching System

- Large and complex software system
  - 10 million lines of code, divided into 50 subsystems
  - Roughly 3,000 developers working at a time
  - Achieved 99.999% reliability
- Multiple releases, multiple versions per release
  - Roughly two years per release
  - At each release, new features added, underlying technology updated
  - Product managers maintained a backlog of features and enhancements

Releases of the 5ESS® Switching System

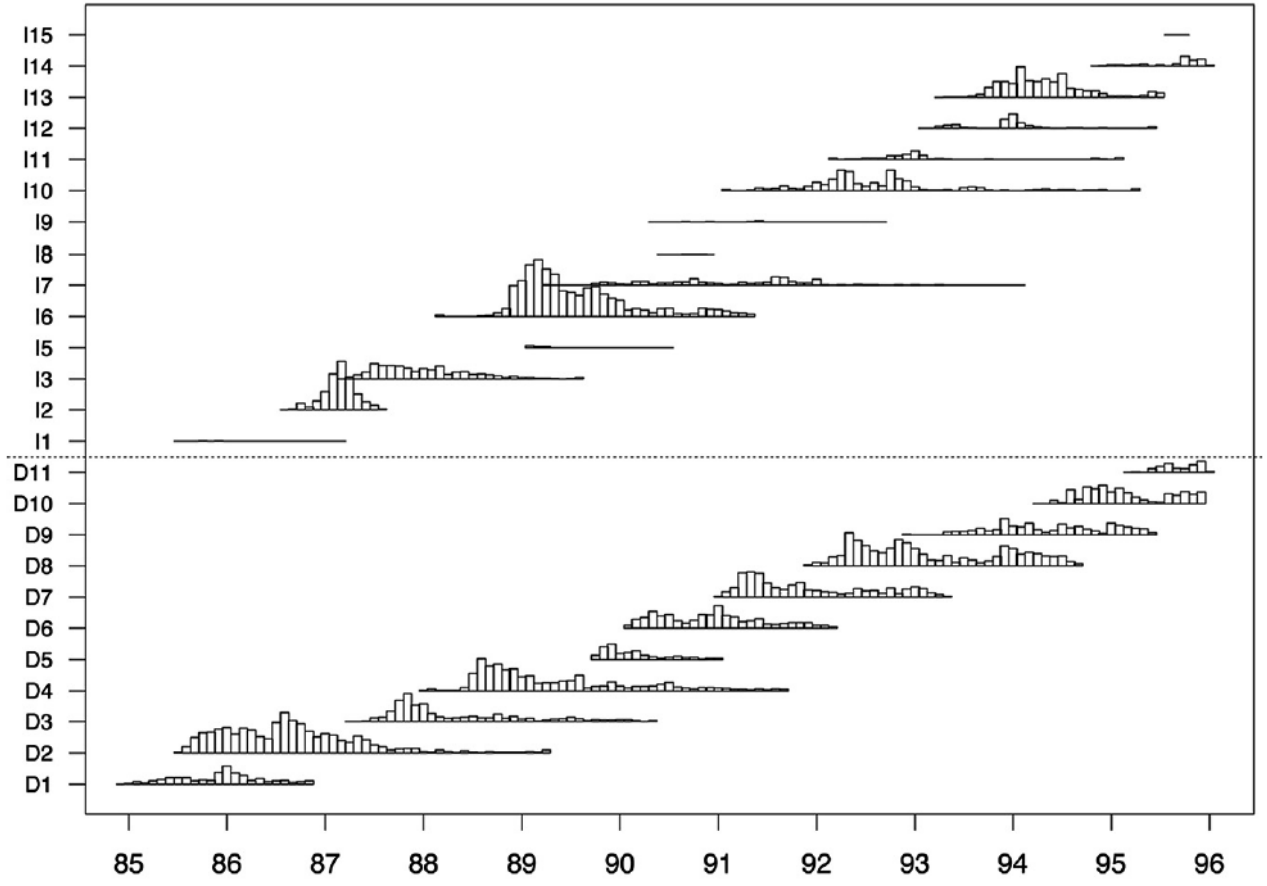
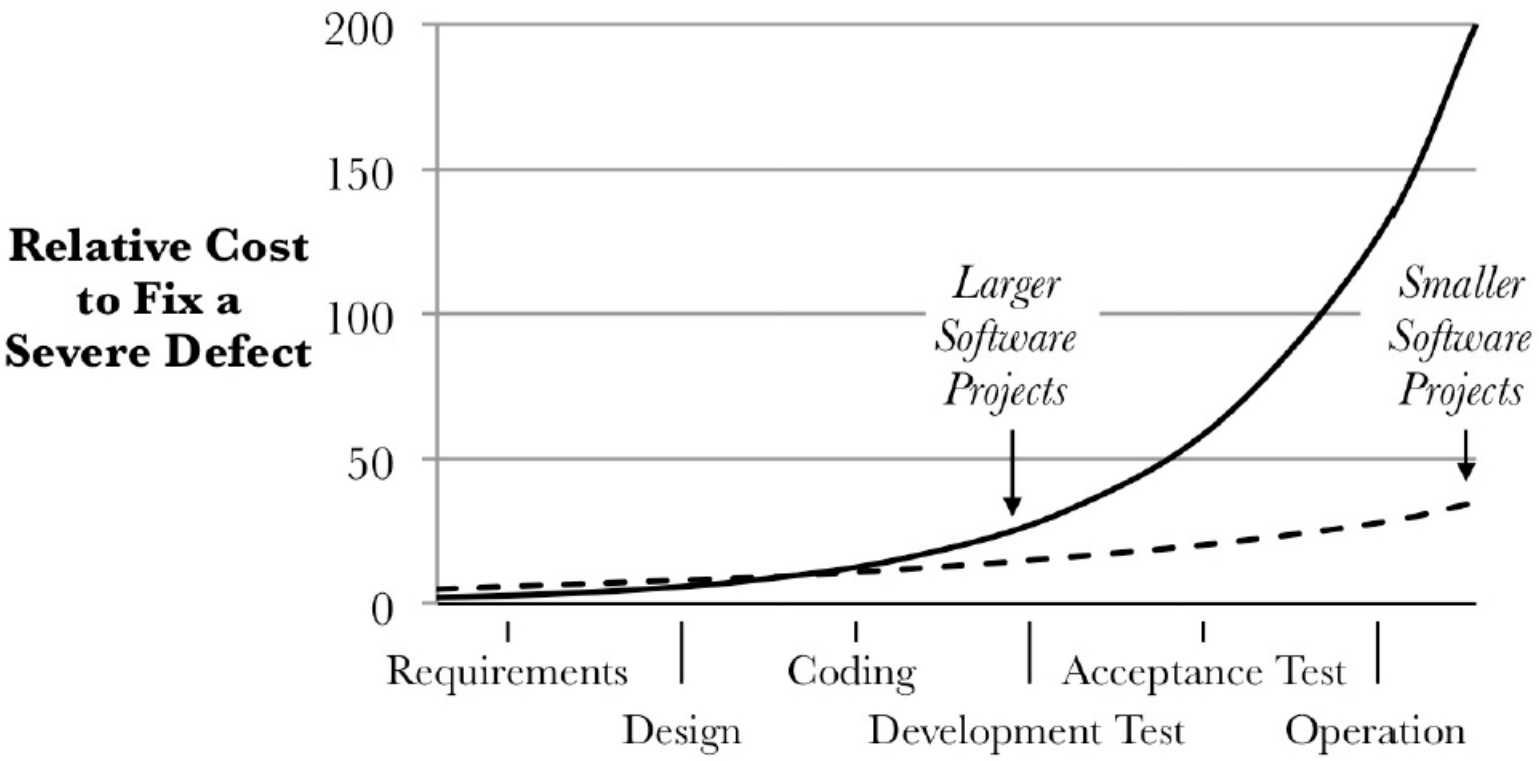


Figure 2.4: Data for releases of the 5ESS® switching system, 1985-1996.

Cost of Change Curve

- What is the cost of changing requirements?
- How hard is it to introduce a change during a project?
- Implications for how much to invest in design and architecture at the start of a project

Cost to Fix a Severe Software Defect



- The later the fix, the greater the cost
- Chart from Boehm [1976,2006]

Cost Ratio for Fixing Software Defects

- 100:1 cost ratio for fixing severe defects during initial requirements versus design and operations in the field
- 2:1 cost ratio for non-severe defects
- Data from a 2002 workshop
  - IBM Rochester: 117:1 increase
    - The increase was 13:1 between coding and testing, 9:1 between testing and operation.
  - Toshiba: 137:1 ratio for time to fix before and after shipment
    - Software factory of 2,600 workers
  - Other organizations reported similar experiences.

Implications of the Cost of Change Curve

Project Size	Severity	Ratio
Large	Severe	~ 100 : 1
Small	Severe	~ 7 : 1
Large	Non-severe	~ 2 : 1

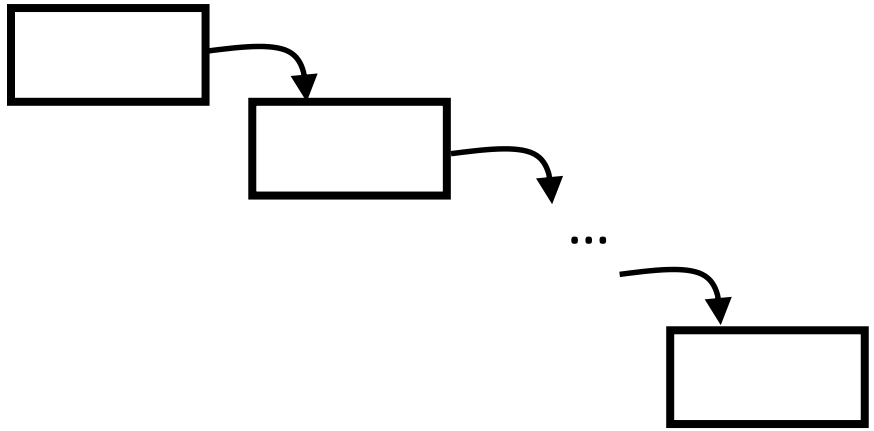
- If the curve is steep (high ratio)
  - then fix earlier
  - motivates plan-driven process
- If the curve is relatively flat (low ratio)
  - then defer changes to allow requirements to settle
  - fundamental assumption behind Agile process

Review: Plan-Driven Processes

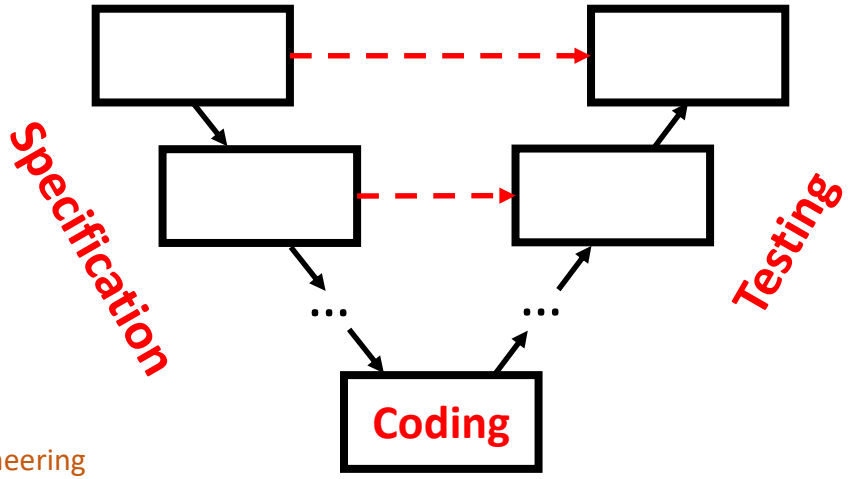
- Definition of plan-driven processes
  - Divide software development into phases, e.g., Requirements, Design, Coding, and Testing
  - May have gates to review whether to go on to the next phase
- Preconditions for success
  - Requirements are relatively stable
  - Phases can be specified up front, before coding / implementation

Review: Plan-Driven Processes

- Waterfall
  - Sequentially through the phases



- V-Processes
  - Test planning along with specification
  - Paired Specification and Testing



Review: Plan-Driven Processes

- In practice, successful projects ensured preconditions before using waterfall (or V)
- Disposable prototype before V-process
  - Example: SAGE in the early 1950s
  - Used prototype to thoroughly understand functions of air defense and the modules and interfaces in a solution

Review: Plan-Driven Processes

- In practice, successful projects ensured preconditions before using waterfall (or V)
- Multiple Releases
  - Example: 5ESS switch
  - Freeze requirements for a release
  - Accommodate changes if they fit in without redesign
  - Extend working code from previous release
- Big difference from iterative
  - Plan each release separately instead of treating a release as an iteration in a larger process

Iterative and Incremental Processes

- Iterative process: sequence of steps or iterations that produce increasingly functional versions of a system
  - Sometimes called iterative and incremental
- Roots from "plan-do-study-act" quality improvement cycles
  - Proposed by Walter Shewhart at Bell Labs in the 1930's
- Came to software through NASA to IBM
  - Used in X-15 hypersonic jet (not a software project)
  - Applied in Project Mercury (which did use software)
  - IBM was a NASA contractor

Iterative and Incremental Processes

- "The basic approach recognizes the futility of separating design, evaluation, and documentation processes in software-system design. The design process is structured by an expanding model seeded by a formal definition of the system, which provides a first, executable, functional model. It is tested and further expanded through a sequence of models, that develop an increasing amount of function and an increasing amount of detail as to how that function is to be executed. Ultimately, the model becomes the system."
  - M.M. Lehman, IBM TJ Watson Research, 1969 (from IBM internal report on development recommendations)

Iterative and Incremental Processes

- "The basic approach recognizes the futility of separating design, evaluation, and documentation processes in software-system design. The design process is structured by an expanding model seeded by a formal definition of the system, which provides a first, executable, functional model. It is tested and further expanded through a sequence of models, that develop an increasing amount of function and an increasing amount of detail as to how that function is to be executed. Ultimately, the model becomes the system."
  - M.M. Lehman, IBM TJ Watson Research, 1969 (from IBM internal report on development recommendations)

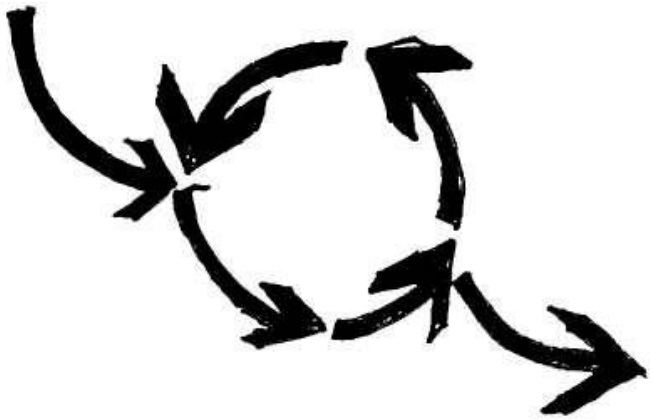
Unix Design Philosophy

- "Make each program do one thing well
  - To do a new job, build fresh rather than complicate old programs …"
- "Design and build software, even operating systems, to be tried early, ideally within weeks.
  - Don't hesitate to throw away the clumsy parts and rebuild them."
- Software utilities "were continually improved by much trial, error, discussion, and redesign."

- Source: Excerpts from a foreword to papers on Unix (McIllroy, Pinson and Tague [1978])

Iterative Processes

- Definition of iterative process
  - Sequence of steps or iterations that produce increasingly functional versions of a system
  - With each iteration, the system does more of what customers want
  - Iteration lengths short and have been getting shorter



Iterative and Incremental Software Processes

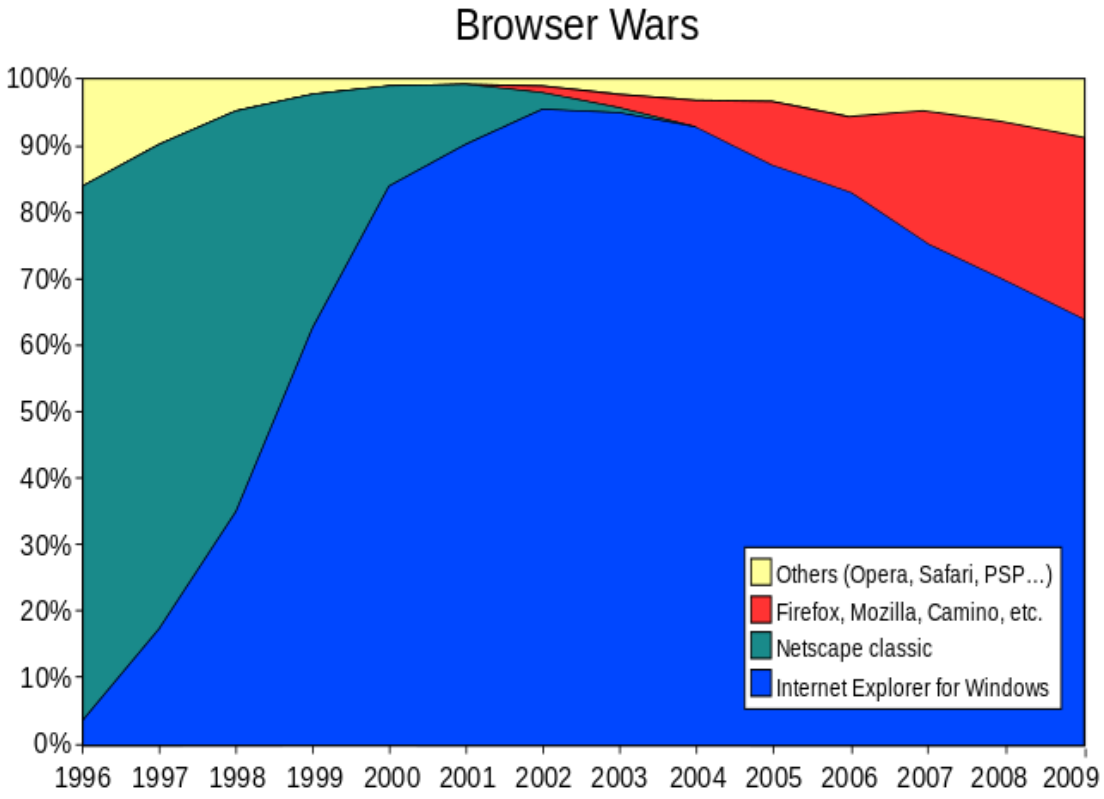
An iteration is a distinct sequence of activities based on an evaluation plan and evaluation criteria, resulting in an executable release

Iterative Processes

- Benefits of iterative (and agile) processes
  - Handle uncertain or dynamically changing requirements
  - Improve design and quality as early users uncover issues
  - Enable parallel development of software, hardware, training, …
- Case studies
  - Netscape Navigator 3.0
  - Space shuttle software (1981)

Case Study: Netscape Navigator 3.0 Browser

- Dominant web browser in the early days of the Internet (mid 1990s)

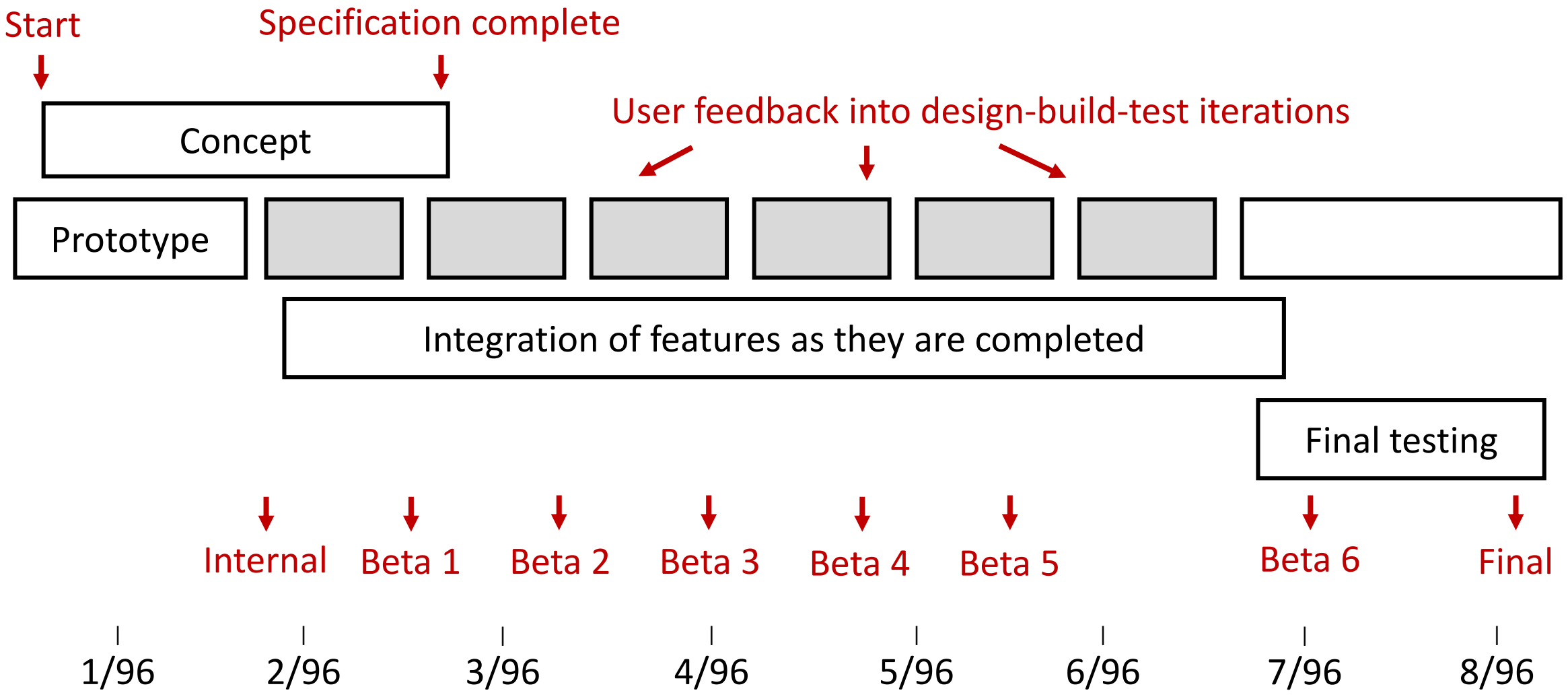


Netscape Navigator 3.0: Quick Iterations

- Uncertain requirements
  - Known unknowns – will customers like the features
- Dynamically changing requirements
  - Unknown unknowns – what will Microsoft do?
- Background: Microsoft missed the Internet "Tidal Wave“[1]
  - In December 1995, launched an all-out effort
  - A “sleeping giant” that had been awakened[2]

1. Gates, William H., III. The Internet Tidal Wave. Internal Microsoft memo. (May 26, 1995)
2. Cusumano, Michael A. and David B. Yoe. Competing on internet Time. The Free Press, New York (1998).

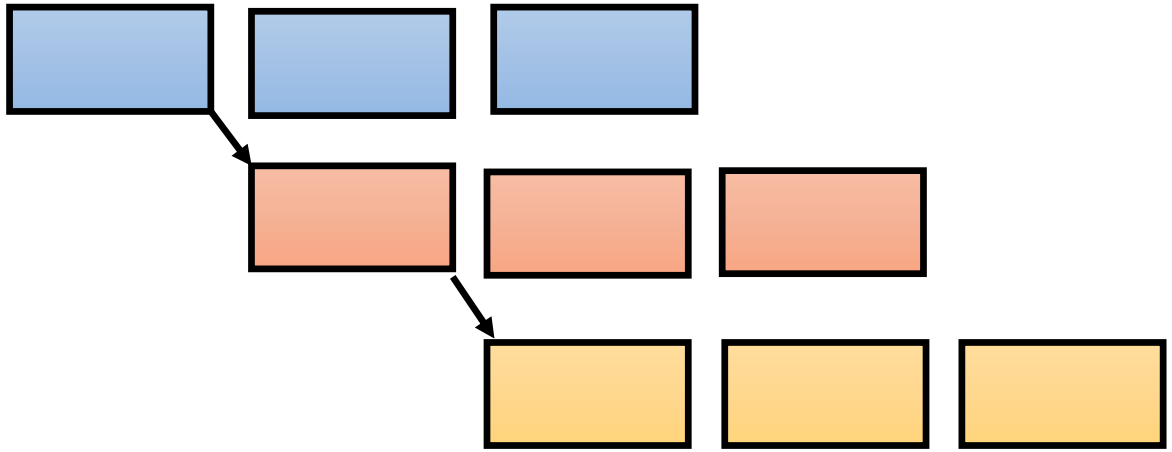
Netscape Navigator 3.0: Iterative Process



Parallel Development of Subsystems

- Sequential: Hardware, then low-level software, then applications

- Iterative Process: enables parallel development



Space Shuttle Software (1981)

- Sequential Process would have been too slow
  - First, develop orbiter hardware
  - Then, develop software for navigation and flight control of orbiter
  - Then, develop simulator and training software
- They were developed in parallel
  - IBM delivered the software iteratively
  - "The first drop for each release represented a basic set of operational capabilities and provided a structure for adding other capabilities on later drops."

Space Shuttle Software (1981)

- Software Process
  - 17 iterations over 31 months
  - Each iteration about 8 weeks
- Changing Requirements
  - Over 2000 requirements changes between 1975 and the first flight in 1981
- Cost of Changing Requirements
  - About $200M spent on software
  - Initial estimate: $20M

Enabling Practices for Iterative Processes

- Study on product development practices:
  - Evaluated 29 software projects from 17 companies (1996-1998)
  - Specifically measured:
    - Product quality (performance, functionality, reliability)
    - Team productivity
- 4 practices correlated with successful projects
  - Early customer feedback to evolve functionality and design
  - Daily builds and automated tests to integrate new code
  - Major investments in a flexible loosely-coupled architecture
  - Experienced team, with experience on diverse projects

Source: MacCormack, Alan. Product-Development Practices that work: How Internet companies build software. MIT Sloan Management Review. 2001.

Agile Manifesto

- "We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
  - Individuals and interactions *over* processes and tools
  - Working software *over* comprehensive documentation
  - Customer collaboration *over* contract negotiation
  - Responding to change *over* following a plan
- That is, while there is value in the items on the right, we value the items on the left more."

Principles Behind the Agile Manifesto

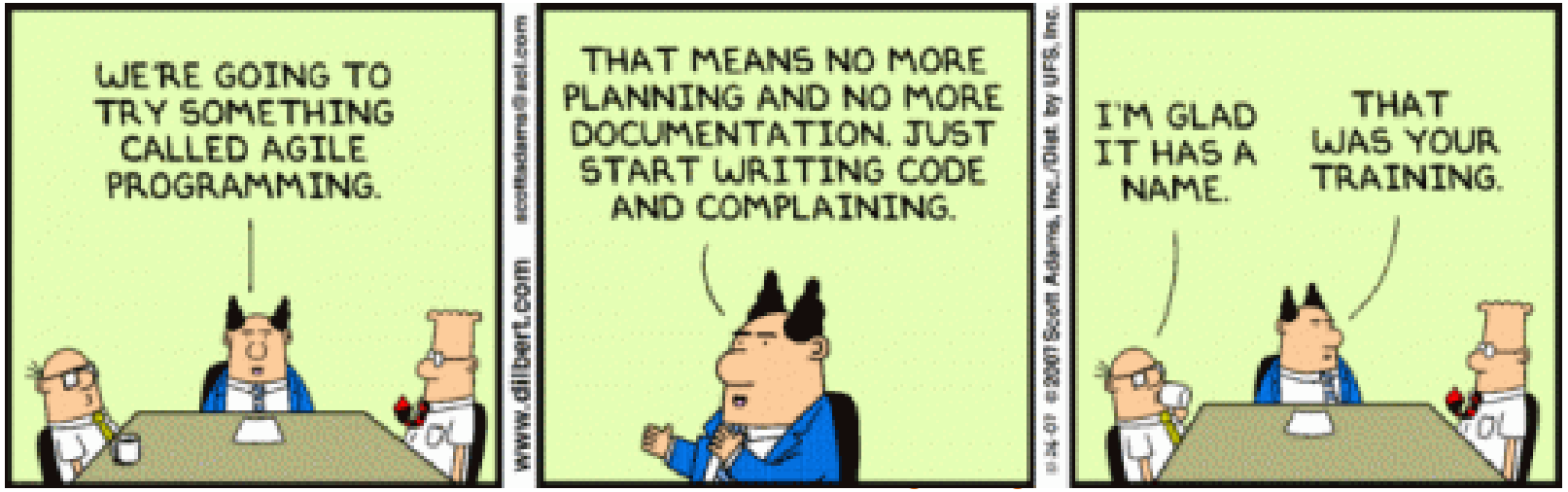
- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Principles Behind the Agile Manifesto

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile Method Emphasis

- Satisfying customers through collaboration
- Delivering working software frequently (in weeks, not months)
- Accommodating changes during development
- Valuing simplicity and technical excellence



Agile Iterations (weeks, not months)

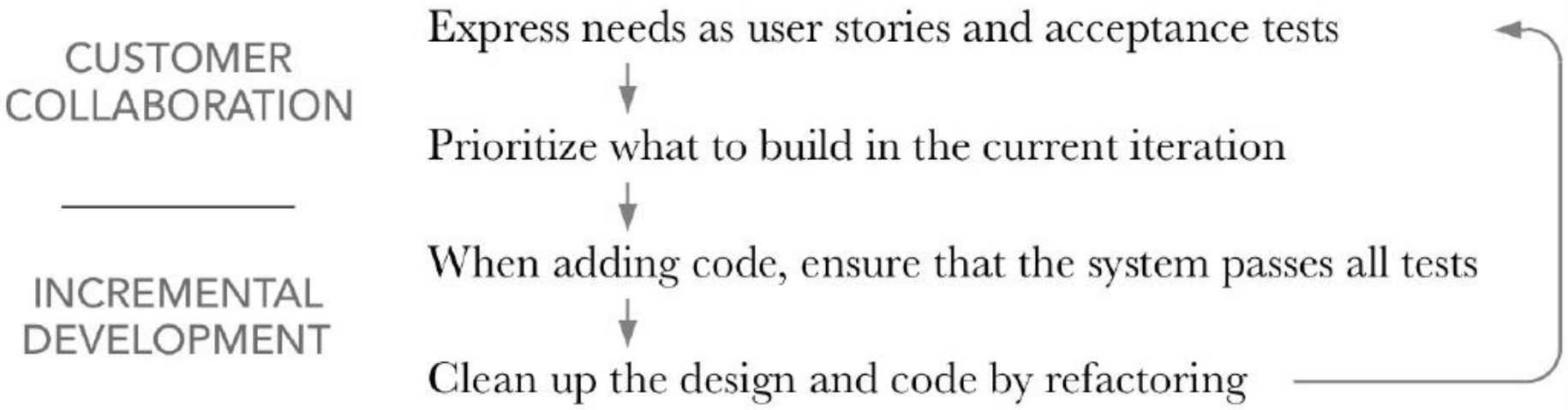


Figure 3.3: Agile iterations.

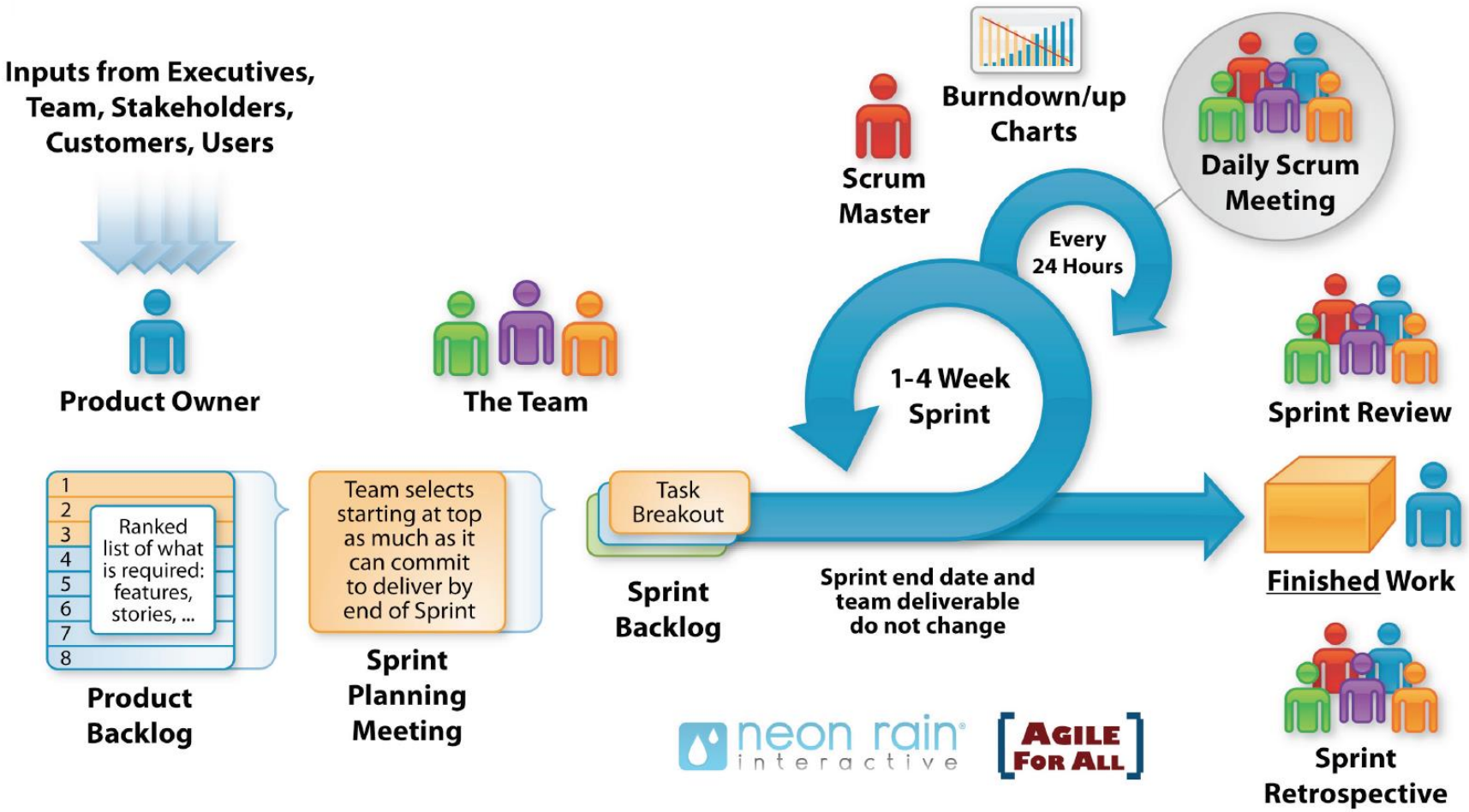
Agile Family of Practices

- Scrum
  - Manage a team's ability to deliver quickly
- Extreme Programming (XP)
  - Software development best practices, taken to "extreme" levels
- Feature-Driven Development
  - Model-driven short iteration process
- Lean Software Development
  - Adapted from lean manufacturing
- …

The Scrum Framework

- Three Roles
  - Product Owner represents stakeholders and the voice of the customer
  - Cross-functional Development Team delivers potentially shippable increments
  - Scrum Master (not a people/project manager) removes impediments
- Events
  - A sprint is a 1-4 week iteration leading to a working product
- Meetings
  - Sprint Planning: select work to be done; prepare the Backlog
  - Daily Scrum: see later slide
  - Sprint Review with stakeholders
  - Sprint Retrospective for continuous improvement

The Scrum Framework



Daily Scrum Meeting

- A Daily Scrum is a project team communication meeting that has specific guidelines:
  - All developers come prepared with their updates for the meeting.
  - The meeting starts precisely on time even if some members are missing.
  - The meeting should happen at the same location and same time every day.
  - The meeting length is set (timeboxed) to 15 minutes.
  - All are welcome, but normally only the core roles speak.
- During the meeting, each team member answers three questions:
  - What have you done since yesterday?
  - What are you planning to do today?
  - Any impediments / stumbling blocks?
    - If so, they are documented / handled by Scrum Master outside the meeting

Extreme Programming Explained (Beck [2000])

- XP takes common-sense principles and practices to extreme levels
  - If code reviews are good, we'll review code all the time (pair programming).
  - If testing is good, everybody will test all the time (unit testing), even the customers (functional testing).
  - If design is good, we'll make it part of everybody's daily business (refactoring).
  - If simplicity is good, we'll always leave the system with the simplest design that supports its current functionality (the simplest thing that could possibly work).
  - If architecture is important, everybody will work defining and refining the architecture all the time (metaphor).
  - If integration testing is important, then we'll integrate and test several times a day (continuous integration).
  - If short iterations are good, we'll make the iterations really, really short—seconds and minutes and hours, not weeks and months and years (the Planning Game).

Agile Practices from Extreme Programming

- User Stories
  - In their words, what the system needs to do for customers
- Acceptance Tests
  - Customer specifies scenarios to test a user story
- Release Planning
  - Together, developers and customers decide on user stories to implement in the next release/iteration
- Test-Driven Development
  - Write tests before implementing each feature
- Refactoring
  - Restructure to clean up the design as new code is added

User Stories: Template and Example

- User Story Template
  - *Feature*: [Name]

    As a [kind of stakeholder]

    I want to [do some task],

    so that [I can achieve some benefit]
- ATM Example
  - *Feature*: Account holder withdraws cash

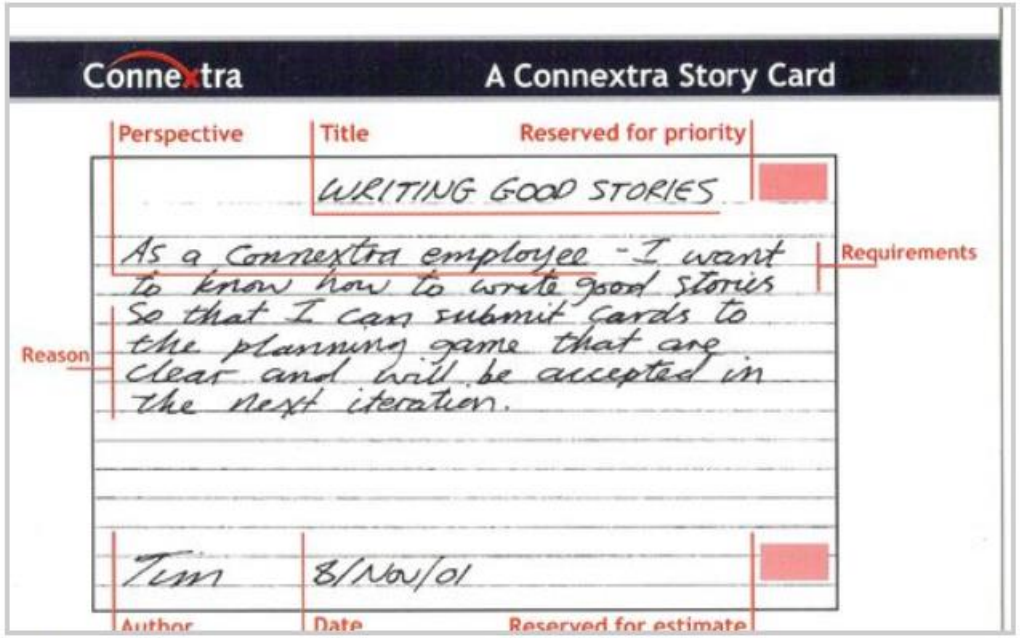
    As a customer

    I want to withdraw cash from an ATM,

    so that I don't have to wait in line at the bank

Structured User Stories

- Keeps stories simple enough to fit on a 3x5 index card
- In addition to the 'As a … I want … so that …' template, the Connextra Story Card included
  - Date
  - Author
  - Customer priority
  - Developer estimate of effort
- The template lives on
  - Connextra, sadly, does not



Connextra, London

Developing a User Story

- Involves multiple people
  - Business stakeholder may need help in framing the narrative
  - The customer may know what they want, but not the cost/benefit
  - A developer can provide a ballpark estimate of what it would take
  - Tester helps define the scope in the form of an acceptance test
- Missing stories
  - If the “want” won’t deliver the “benefit” you may have a missing story
  - If a story is too complex to fit into an iteration, break it down into multiple stories
- A spike is an investigation
  - If the developers don’t see how to even make a ballpark estimate, they may need a spike to understand the customer requirements

What Makes a Good User Story?

- Make user stories SMART, where SMART stands for
  - *Specific*
  - *Measurable*
  - *Achievable*
  - *Relevant*
  - *Time-bound*
- Minimum Viable Product
  - Subset of the full set of user stories that would make for a viable product

Source: Wake, Bill. INVEST in Good Stories and SMART tasks. Exploring Extreme Programming (website). 2003.

Acceptance Criteria Template for User Stories

- *Given* some initial context (the givens or preconditions),
  - *And* some more context, …
- *When* an event occurs,
- *Then* ensure some outcome
  - *And* another outcome …
- Not all customer scenarios are this simple
  - May need a sequence of “thens” and “whens”; e.g., with menus

Acceptance Tests Should be Executable

- ATM Scenario 1: Account is in credit
  - *Given* the account is in credit
    - *And* the card is valid
    - *And* the dispenser contains cash
- *When* the customer requests cash
- *Then* ensure the account is debited
  - *And* ensure cash is dispensed
  - *And* ensure the card is returned

Acceptance Tests Should be Executable

- ATM Scenario 2: Account is overdrawn past the limit
  - *Given* the account is overdrawn
    - *And* the card is valid
- *When* the customer requests cash
- *Then* ensure a rejection message is displayed
  - *And* ensure cash is not dispensed
  - *And* ensure the card is returned

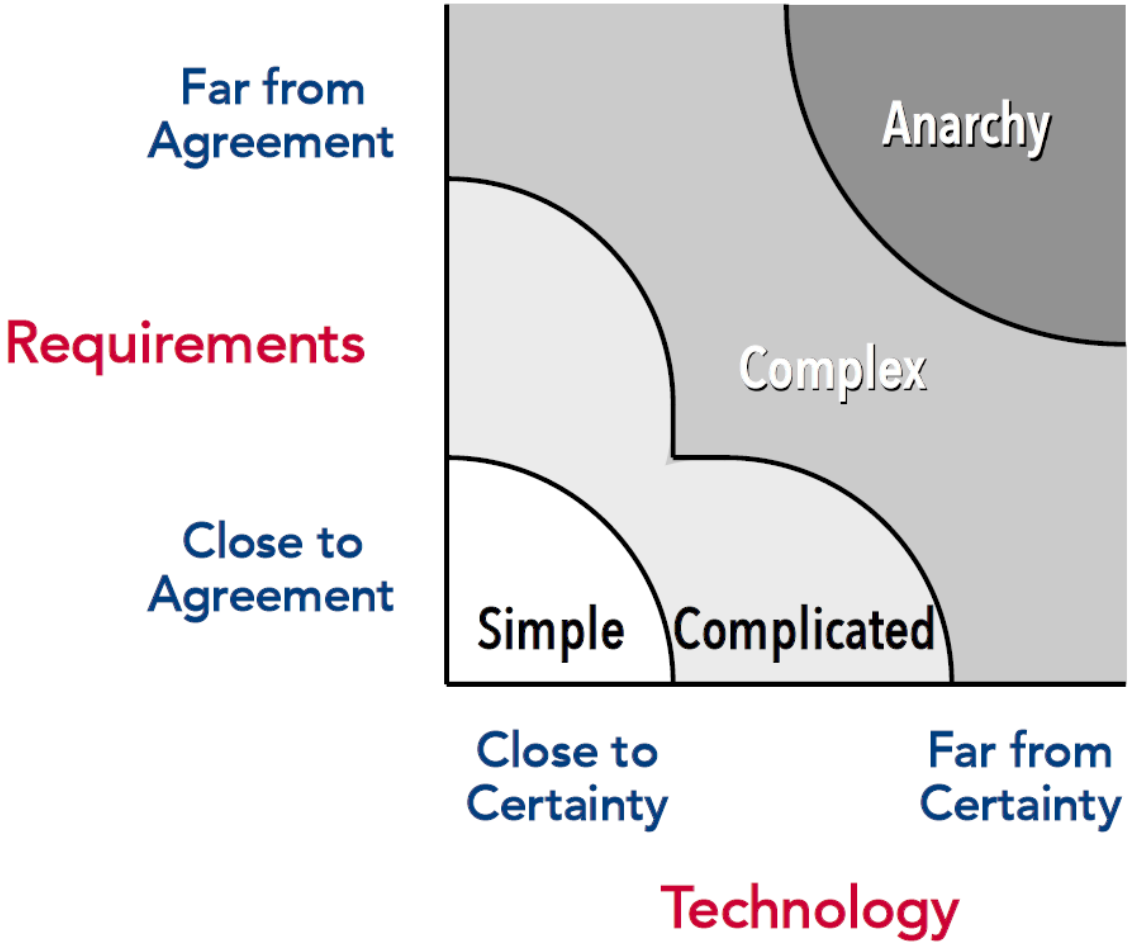
User Stories: Rough Estimates of Effort

- Customer owns the priorities, developer owns the cost estimates
- Group stories by levels of effort
  - One point: I know exactly how to do this, and can do it in half a day.
  - Two points: I know exactly how to do this, but it will be some work.
  - Three points: "Somehow we will implement this feature."
- Three points almost always turns into more
  - It is a red flag that the story needs to be broken into smaller stories.
- Velocity: average number of points completed each iteration

Complexity in Development Projects

- For user stories
  - Simple
    - 1 point
  - Complicated
    - 2 – 3 points
  - Complex
    - Break down the story
  - Anarchy
    - Keep talking to clarify customer needs and goals

- Source: Ogunnaike and Ray [1992]



Pivotal Tracker

- A story-based project planning tool
- Recommended video
  - https://www.youtube.com/watch?v=mTYcHg51sWY

Limitations of User Stories and Backlogs

- Lack context for user actions
  - When the user is doing something, what is their larger goal
- Unsure of completeness
  - The stories can keep increasing, seemingly without bound
- Lack lookahead at the difficulty of upcoming work (in practice)
  - Variations and exceptions may be discovered too late

Test-Driven Development

- The "practice of test-first development, planning and writing tests before each micro-increment" was used as early as NASA's Project Mercury, in the early 1960s
  - Larman and Basili [2003]

Test Driven Development

- Each new feature begins with writing a test
  - Base the test on requirements; e.g., user stories, use case, exceptions
  - Test what the feature should do
- Run all tests and see if the new one fails
  - Does the new test fail for the expected reason?
  - If the test succeeds, either the feature exists, or the test is defective
- Write just enough code to pass the test
  - The code can be inelegant; it will get cleaned up in a later step
- Run all tests
  - If all tests now pass, the code meets the tested requirements
- Refactor code as necessary
  - See next slide

Test Driven Development

- Refactor code as necessary
  - Remove any duplication
  - Move code from where it was added to pass test to where it belongs
  - Does the code reflect the developer's intent?
  - Do the variable and method names reflect their current usage?
  - Minimize the number of classes and methods
  - Refactor tests as well; tests are part of the maintenance overhead
- Repeat
  - Add another test that adds functionality
  - Keep steps small, a few edits at a time
  - If the new code does not rapidly satisfy the new test or if other tests fail unexpectedly, undo and revert – avoid excessive debugging

Test Driven Development: Shortcomings

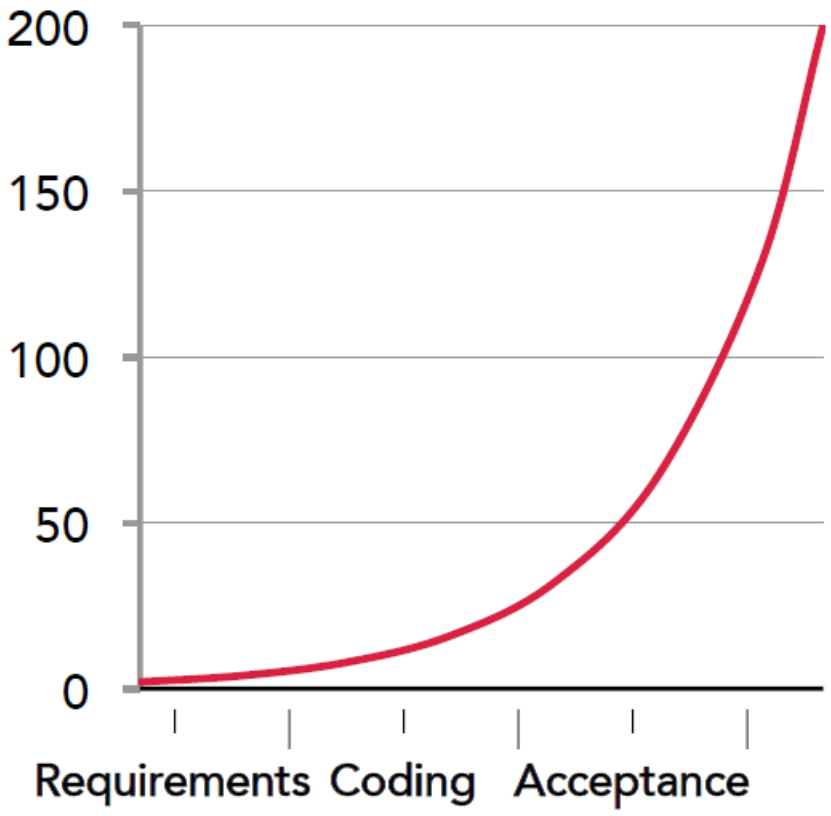
- Completeness
  - Ensuring inclusion of tests that should fail
  - The tests may not handle worst cases and exception conditions
  - The tests and the code share the developer's blind spots
- The code passes the test, but is it good enough?
  - Were performance, scalability – the 'ilities – considered
- A late design change can cause many tests to fail
  - The failing tests will need to be fixed individually

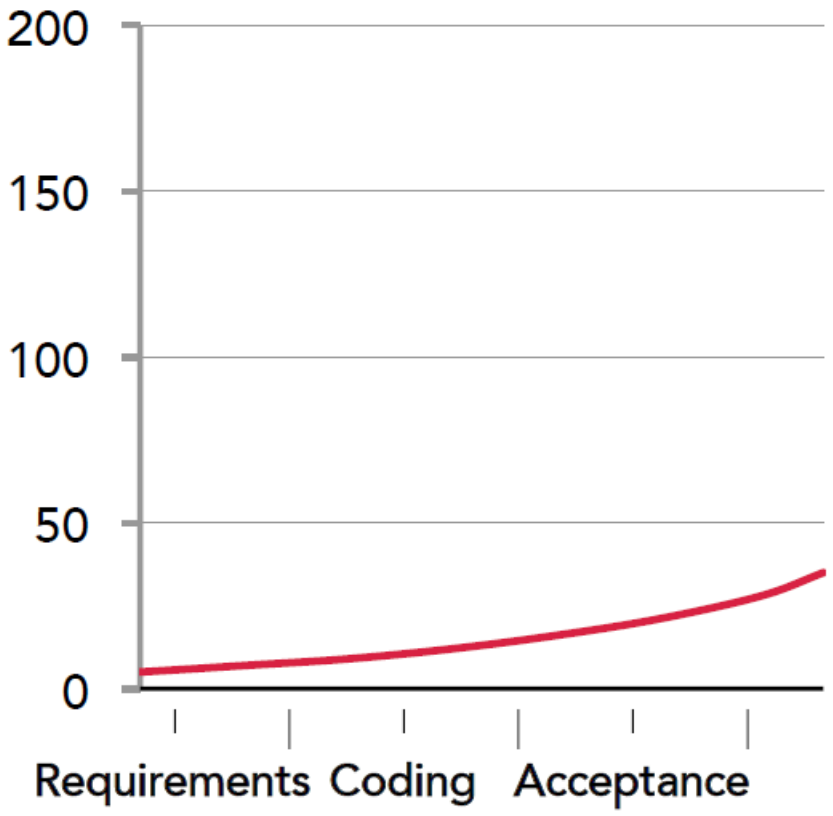
When to Design? It's When Not Whether

- Flexible design early to set direction
  - Start with a design that is flexible enough to handle later changes without major rework
- Design incrementally
  - Start with a minimal plausible design, add and refactor at the end of each iteration
- The difference is in the point of view
  - How much to invest in design for flexibility?
  - How much to invest in design to get a plausible design?
  - Depends on the project

When to Design?

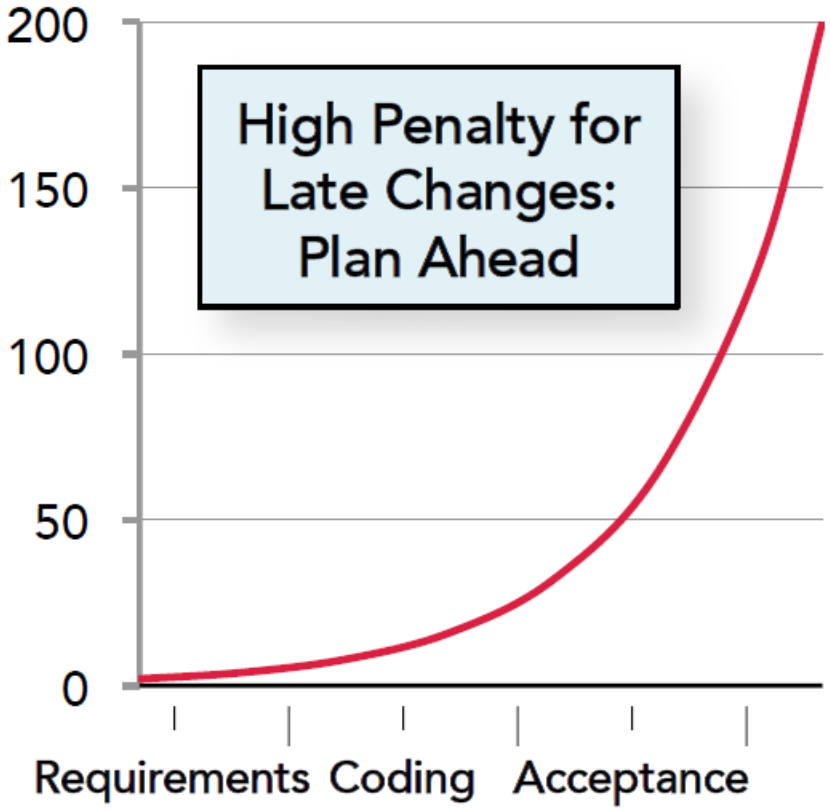
- What is the shape of the cost of change curve?

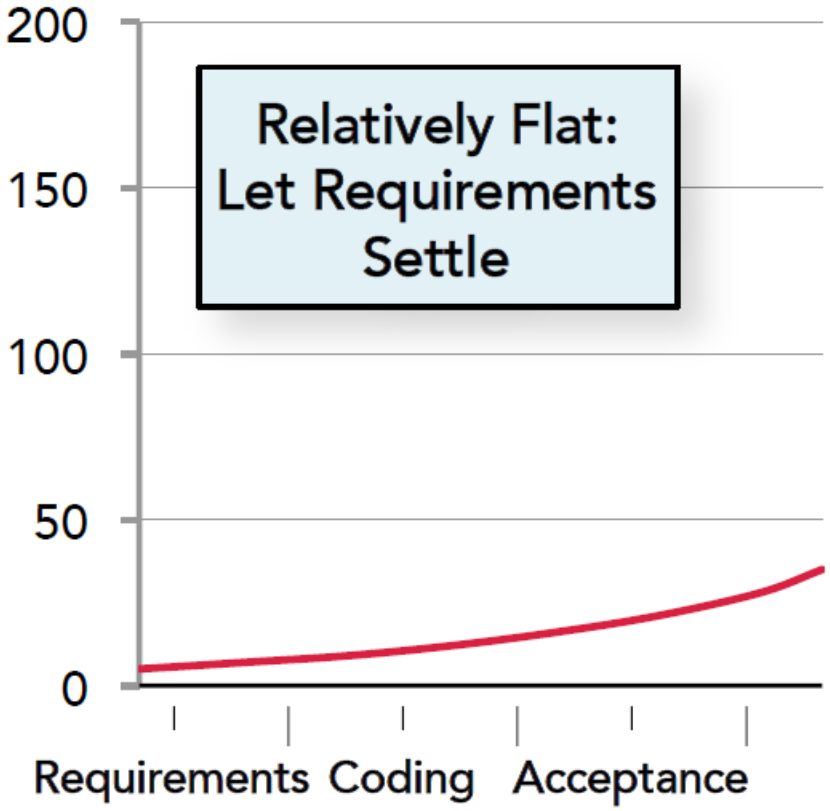




When to Design?

- What is the shape of the cost of change curve?





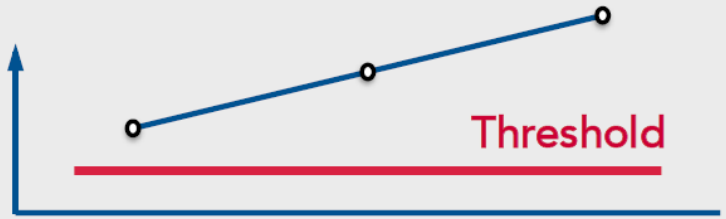
When to Design? XP Misinterpreted

- Early Interpretation
  - YAGNI means no design
  - Martin Fowler article entitled, “Is design dead?”
    - “Evolutionary design is a disaster”

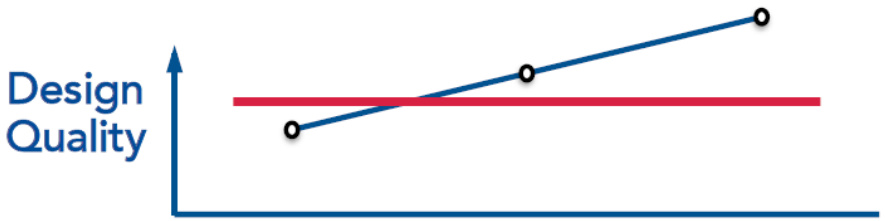
- From Extreme Programming Explained, 2nd edition (2005)
  - “The advice to XP teams is not to minimize design investment over the short run, but to keep the design investment in proportion to the needs of the system so far. The question is not whether or not to design, the question is when to design.”

When to Design? Depends on the Task

- With a low Threshold, any design will do



- Some thought or experience is necessary



- No amount of pure thought will suffice

Source: Beck and Andres [2005]

When to Design? Depends on the Task

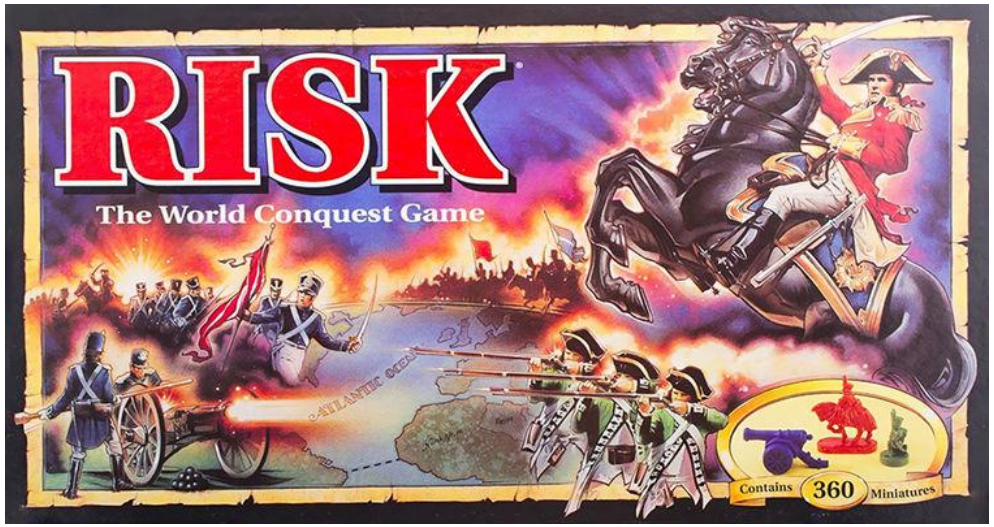
- Experience adds little with this task

- Experience really helps with this task

Source: Beck and Andres [2005]

Risk Reduction

- Factors that contribute to success or failure can be thought of in terms of *risk*
- Choosing the right process can mean the difference between success or failure



Assessing Project Risk: Sample Questions

Requirements

Is the application critical or discretionary?
Are the requirements fixed or fluid?

Development

Is the team experienced?
Is it colocated or distributed?

Process?

Technology

Is the architecture flexible?
Are the tools proven or risky?

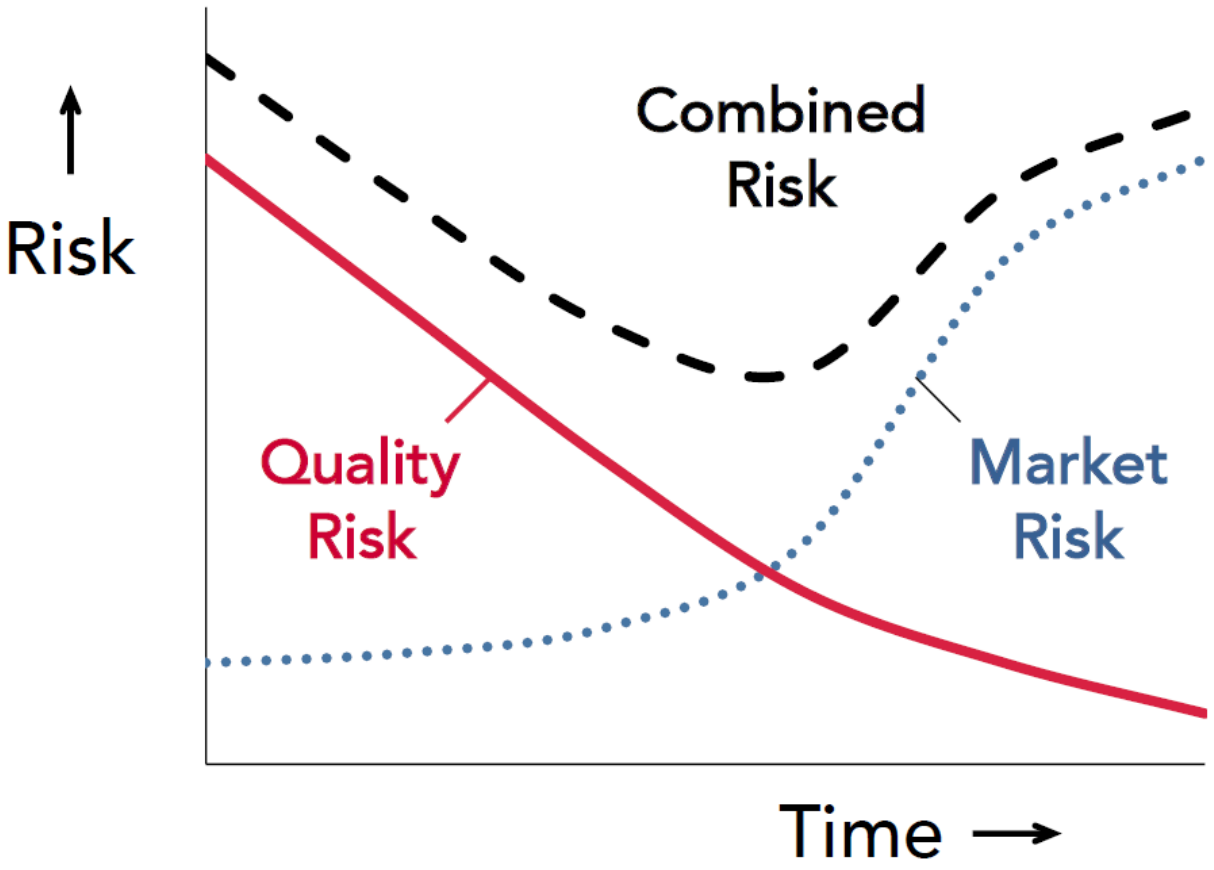
Constraints

Is the priority cost, schedule, or quality?
Does the system have to comply with regulations or standards?

Example: Netscape Communicator 4.0

- Netscape successfully used “Iterative” for Navigator 3.0
  - Discussed earlier
- Very next project Communicator 4.0 had poor quality
  - Same or essentially the same Iterative Process
- Communicator 4.0 had multiple risks
  - Requirements changes: email features changed along the way
  - Unproven groupware features – not embraced by customers
  - Existing code from Navigator 3.0 needed to be re-architected
  - Chose Java for multi-platform – Java was new; performance suffered
  - Skills shortage – didn’t have enough testers for Windows, Mac, Unix

Combination of Risks



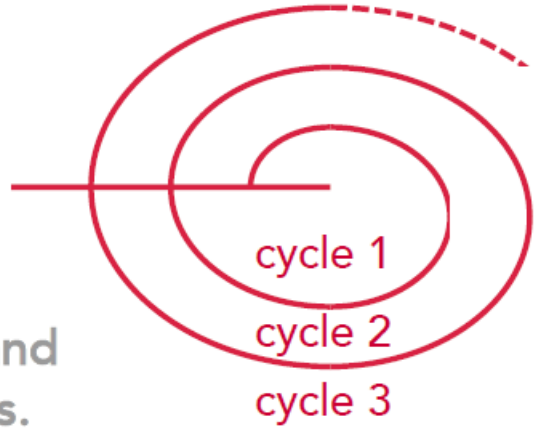
Spiral Risk Reduction Framework

Objectives
Constraints
Alternatives

2. Identify stakeholders' win conditions

3. Reconcile win conditions. Establish next level objectives, alternatives, constraints

1. Identify next level stakeholders.



4. Evaluate product and process alternatives. Resolve risks

7. Review. Commit.
6. Validate product and process definitions.

5. Define next level of product and process, including partitions.

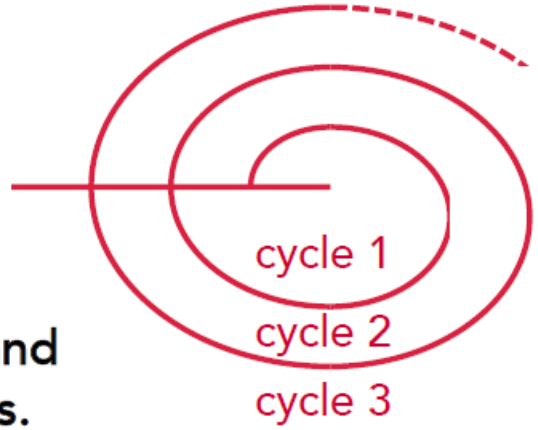
Risks
Reviews
Commitment

Source: Boehm 1986

Spiral Risk Reduction Framework

2. Identify stakeholders' win conditions

1. Identify next level stakeholders.



3. Reconcile win conditions. Establish next level objectives, alternatives, constraints

7. Review. Commit.

6. Validate product and process definitions.

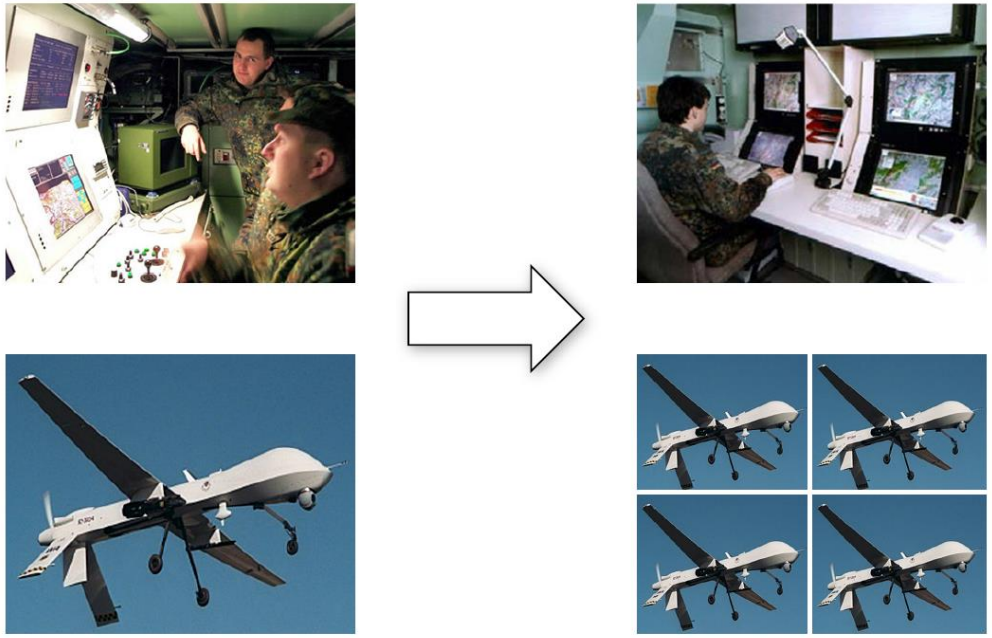
4. Evaluate product and process alternatives. Resolve risks

5. Define next level of product and process, including partitions.

Source: Boehm 1986

Scaling Remote Controlled Operations

- Problem: reduce number of pilots necessary to control drones
  - Instead of 2 people for each drone, build software that enables 1 person to control 4 drones



Source: Boehm [2013]

Scaling Remote Controlled Operations

- 1st cycle:
  - 4 competing teams each awarded $5M, $5M for evaluation
  - Review concludes 4:1 ratio is not realistic
- 2nd cycle:
  - 3 competing teams each awarded $20M, $15M for evaluation
  - Review concludes 1:1 ratio is possible
- 3rd cycle:
  - 1 team selected to build a viable system
  - Achieved a ratio of 1:1 (twofold productivity improvement)

Source: Boehm [2013]

Process Conclusions

- Fit the software development process to the project
- Recommendations for processes
  - Deep customer involvement
  - Time-boxed iterations
  - Ensure software works frequently
- Variation within a process
  - When to design
  - How much to test
  - Balance of scope, schedule, and cost
  - Risk assessments help guide decisions about processes