Software Engineering

Dr. Young-Woo Kwon

SOFTWARE DEV. LIFECYCLE

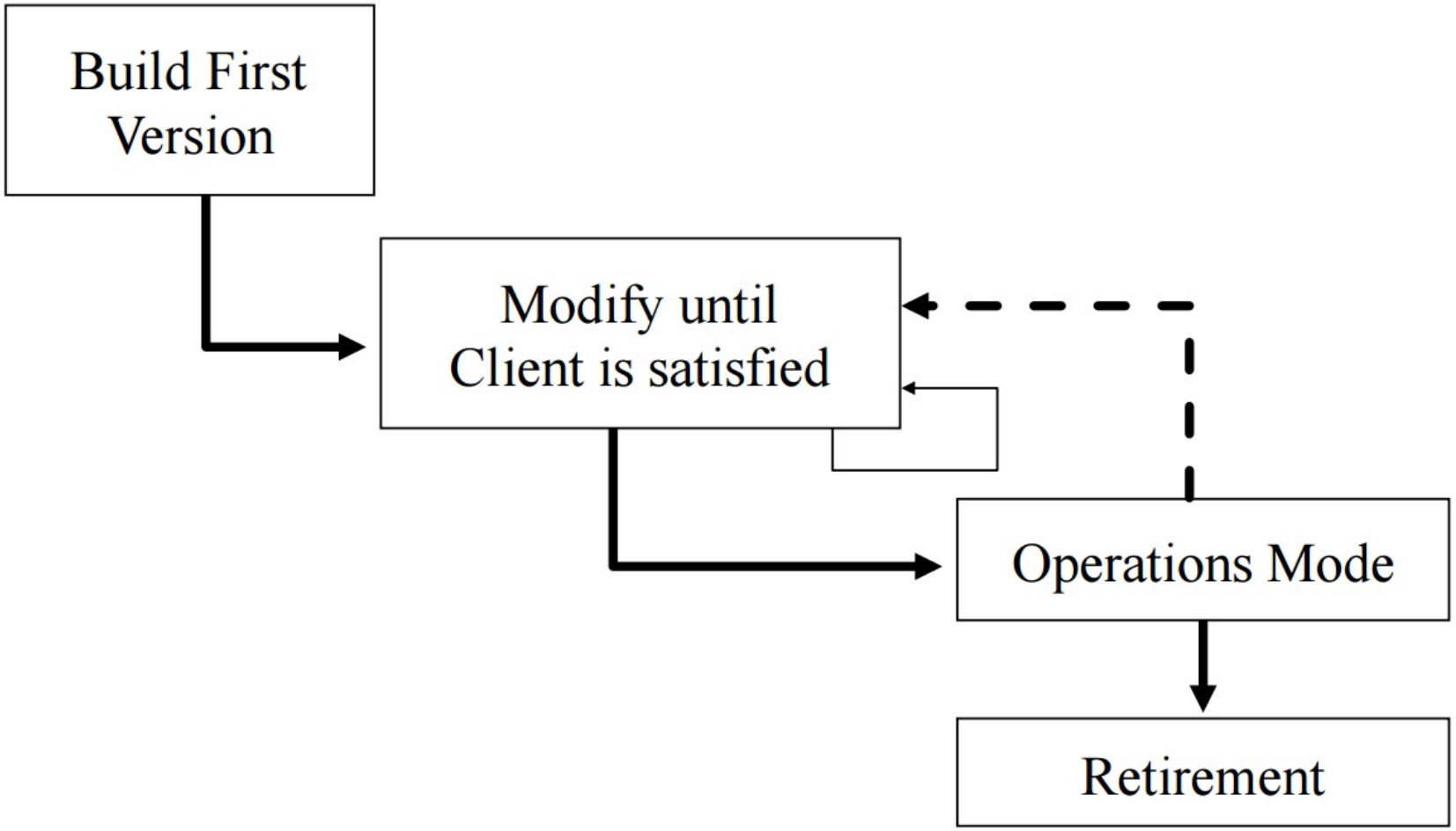
Software Development Lifecycle

- Anti Lifecycle
  - Code & Fix
- Exemplars
  - Waterfall
  - Rapid Prototyping
  - Incremental
  - Spiral
  - Agile

Code and Fix

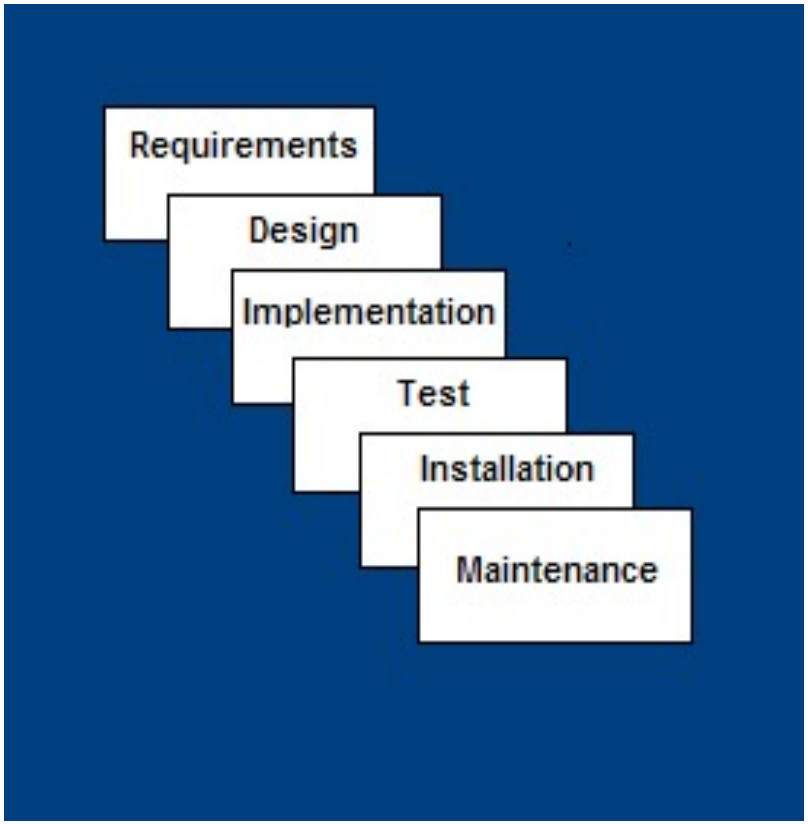
- Useful for small-scale, personal development
- Problems become apparent in any serious coding effort
  - No process for things like versioning, testing, change management, etc.
  - Difficult to coordinate activities of multiple programmers
  - Non-technical users cannot explain how the program should work
  - Programmers do not know or understand user needs

Code and Fix


Build First Version
Modify until Client is satisfied
Operations Mode
Retirement

Waterfall Model

- Proposed in early 70s by Winston Royce



- **Requirements** – defines needed information, function, behavior, performance and interfaces.
- **Design** – data structures, software architecture, interface representations, algorithmic details.
- **Implementation** – source code, database, user documentation, testing.

Strengths of the Waterfall Model

- Easy to understand, easy to use
- Provides structure to inexperienced staff
- Milestones are well understood
- Sets requirements stability
- Good for management control (plan, staff, track)
- Works well when quality is more important than cost or schedule

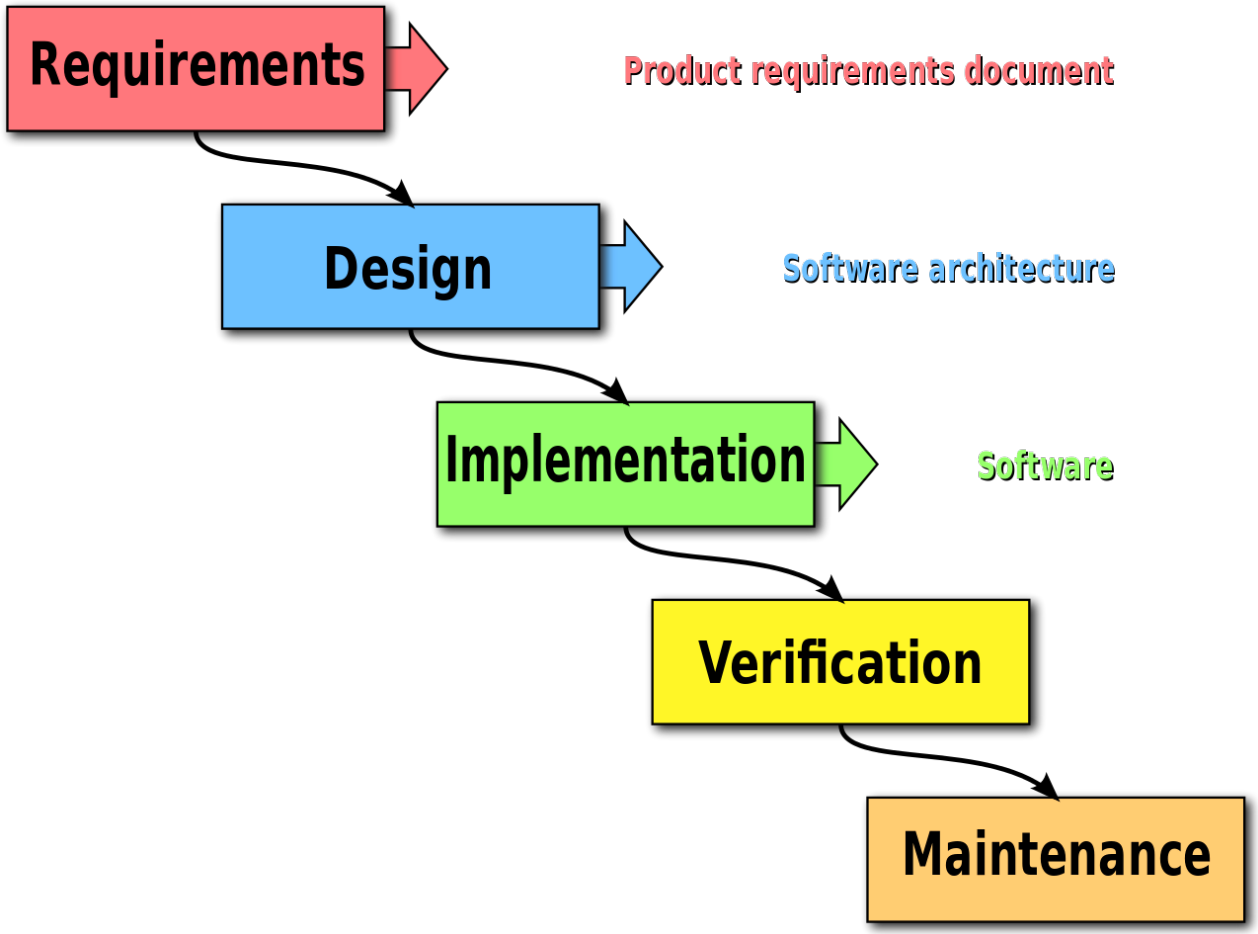
Waterfall Deficiencies

- All requirements must be known upfront
- Deliverables created for each phase are considered frozen – inhibits flexibility
- Can give a false impression of progress
- Does not reflect problem-solving nature of software development – iterations of phases
- Integration is one big bang at the end
- Little opportunity for customer to preview the system (until it may be too late)

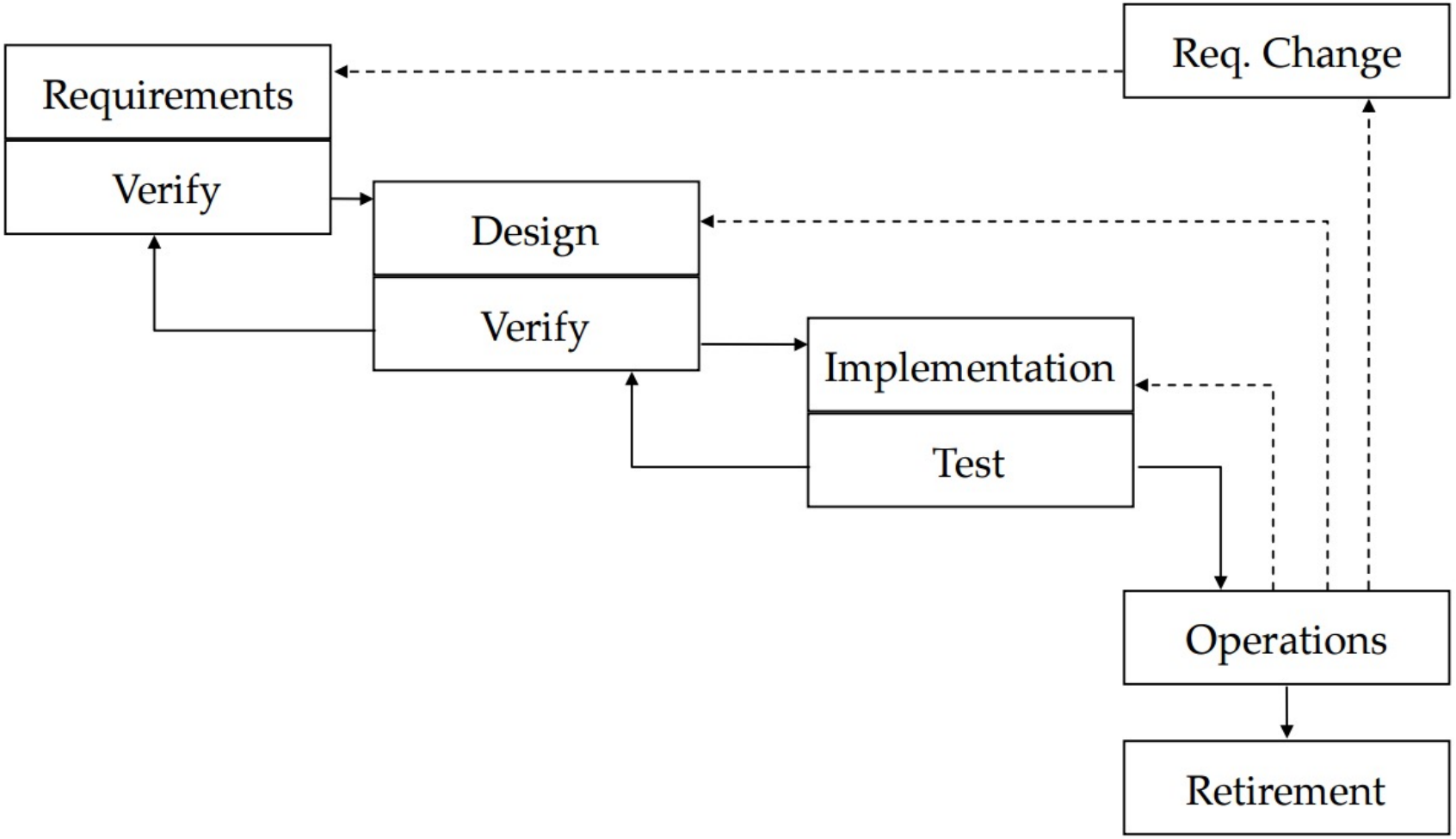
When to use the Waterfall Model

- Requirements are very well known
- Product definition is stable
- Technology is understood
- New version of an existing product
- Porting an existing product to a new platform.

Waterfall


Requirements
Product requirements document
Design
Software architecture
Implementation
Software
Verification
Maintenance

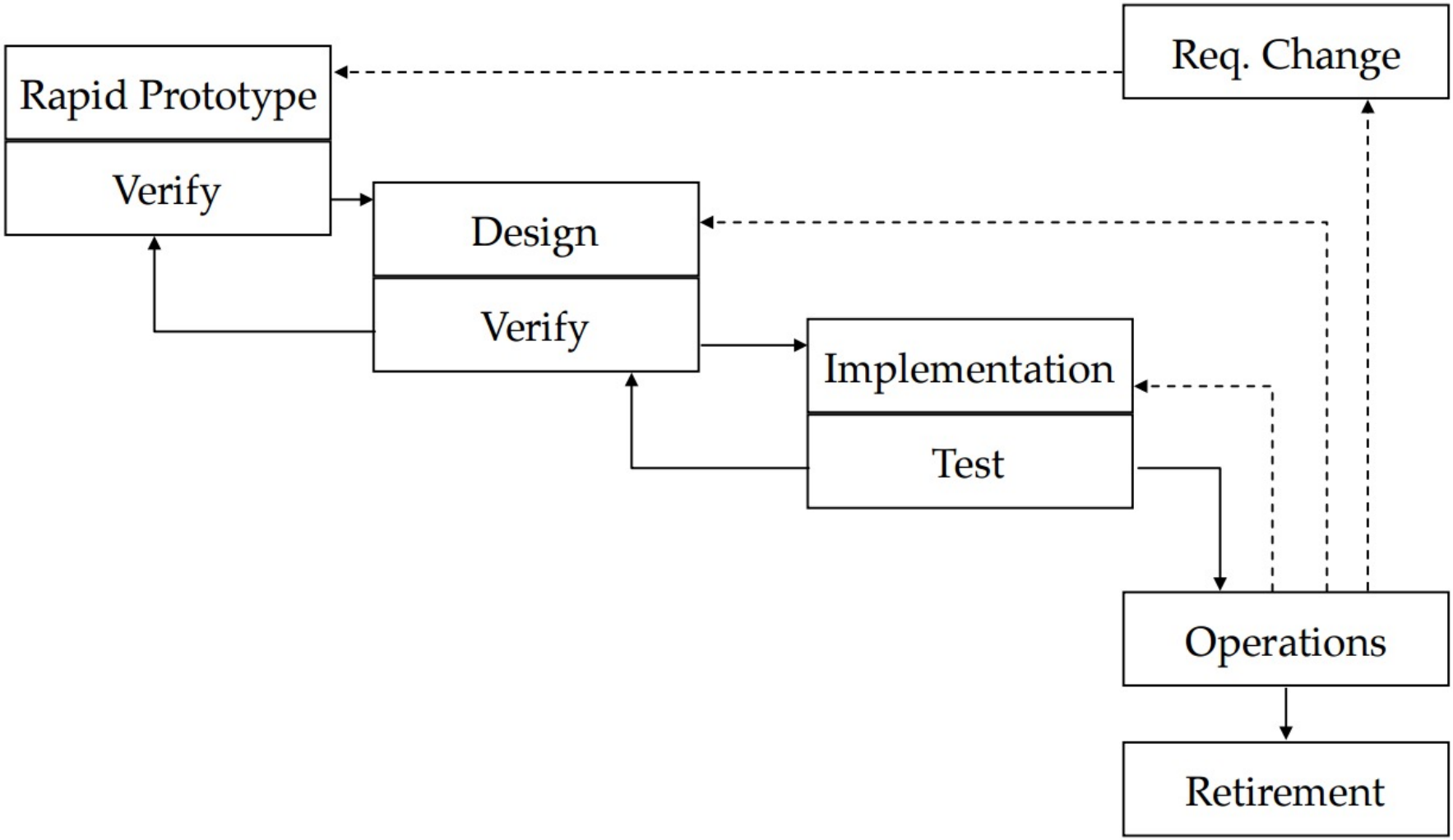
Waterfall with Feedback



Rapid Prototyping

- Prototypes are used to develop **requirements specifications**
  - Once requirements are known, waterfall is used
- Prototypes are discarded once design begins
  - Prototypes should not be used as a basis for implementation
  - Prototyping tools do not create production quality code
- In addition, customer needs to be "*educated*" about prototypes
  - They need to know that prototypes are used to answer requirements related questions
  - Otherwise, they get impatient

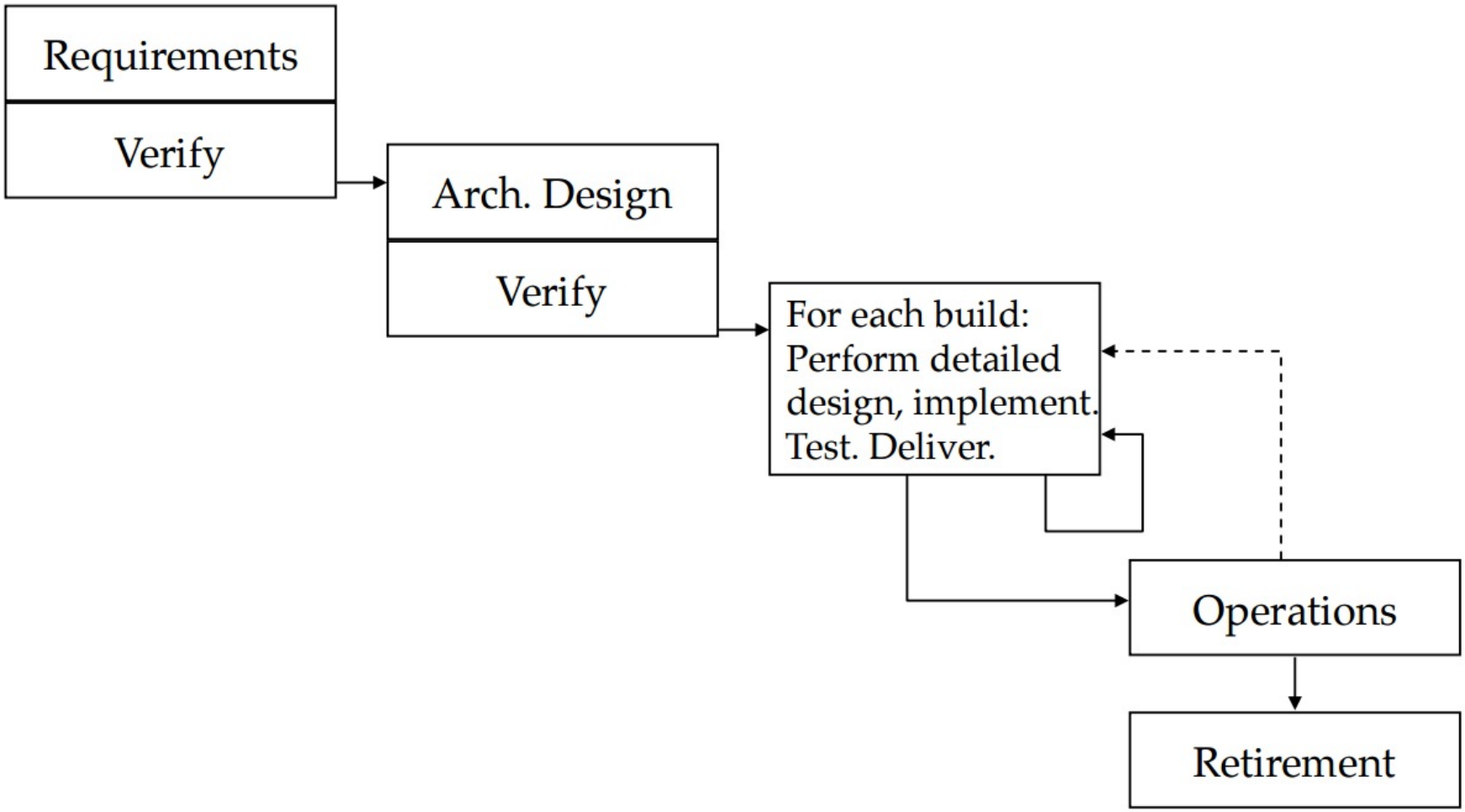
Rapid Prototyping



Incremental

- Used by Microsoft (at least when building Windows XP)
  - Programs are built everyday by the build manager
- Iterations are planned according to features
  - Feature1 and 2 are being worked on in iteration 1
  - Feature 3 and 4 are in iteration 2

Incremental



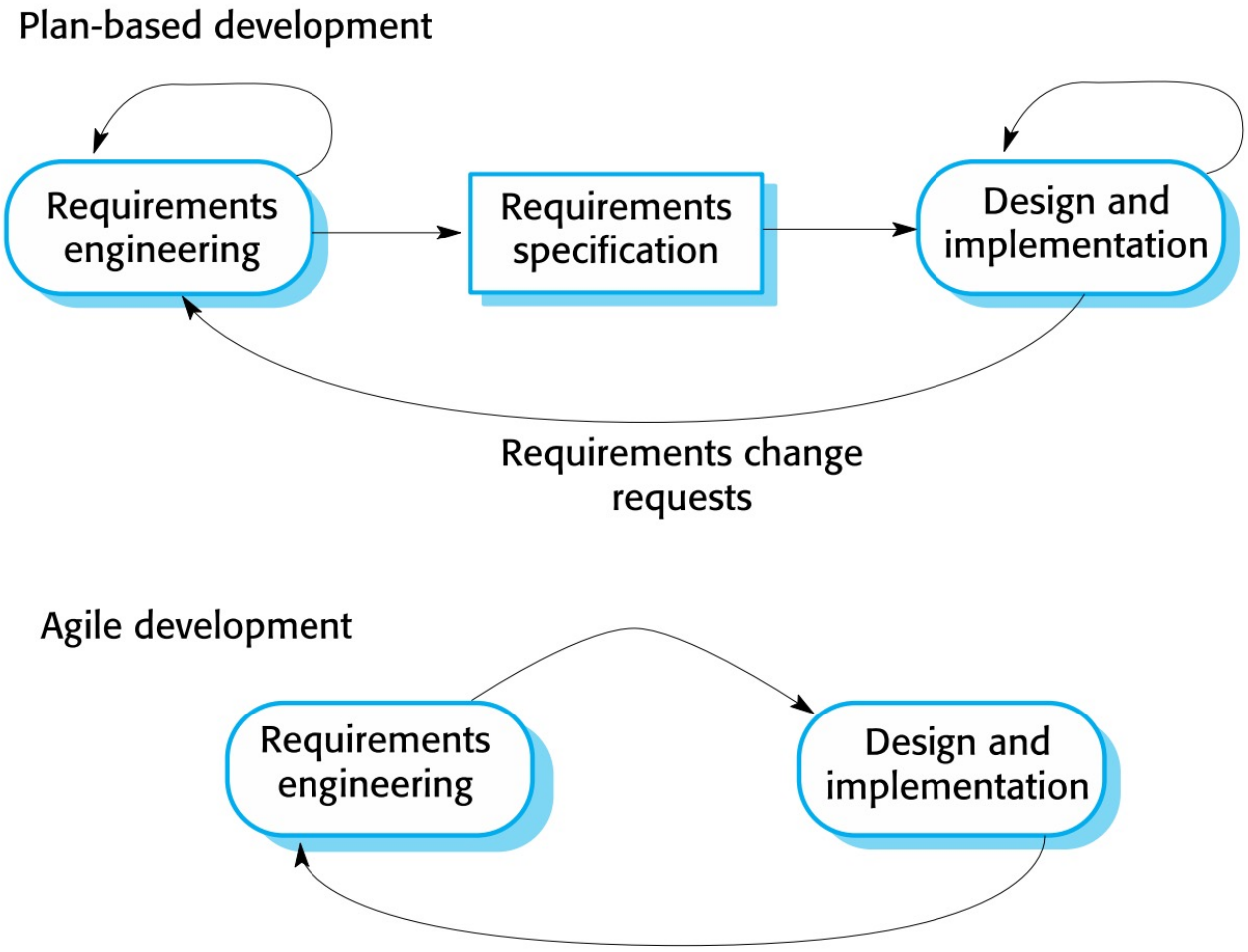
Break

AGILE SOFTWARE DEVELOPMENT

Agile

- Agile development is a response to the problems of traditional "*heavyweight*" software development processes
  - Too many artifacts
  - Too much documentation
  - Inflexible plans
  - Late, over budget, and buggy software

Plan-driven vs. Agile development



Plan-driven vs. Agile development

- Plan-driven development
  - A plan-driven approach is based around separate development stages with the outputs to be produced at each of these stages planned in advance.
- Agile development
  - Specification, design, implementation and testing are interleaved and the outputs are decided through a process of negotiation during the software development process.

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.

12 Principles behind the Agile Manifesto

1. Our highest priority is to **satisfy the customer** through early and **continuous delivery** of valuable software.
2. **Welcome changing requirements**, even late in development. Agile processes harness change for the customer's competitive advantage.
3. **Deliver working software frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

12 Principles behind the Agile Manifesto

4. Business people and developers must **work together daily** throughout the project.
5. Build projects around **motivated individuals**. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is **face-to-face conversation**.

12 Principles behind the Agile Manifesto

7. **Working software** is the primary measure of **progress**.
8. Agile processes promote **sustainable development**. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to **technical excellence and good design** enhances agility.

12 Principles behind the Agile Manifesto

10. **Simplicity** is essential.
11. The best architectures, requirements, and designs emerge from **self-organizing teams**.
12. At regular intervals, the team reflects on how to become more effective, then **tunes and adjusts its behavior** accordingly.

Agile Methods

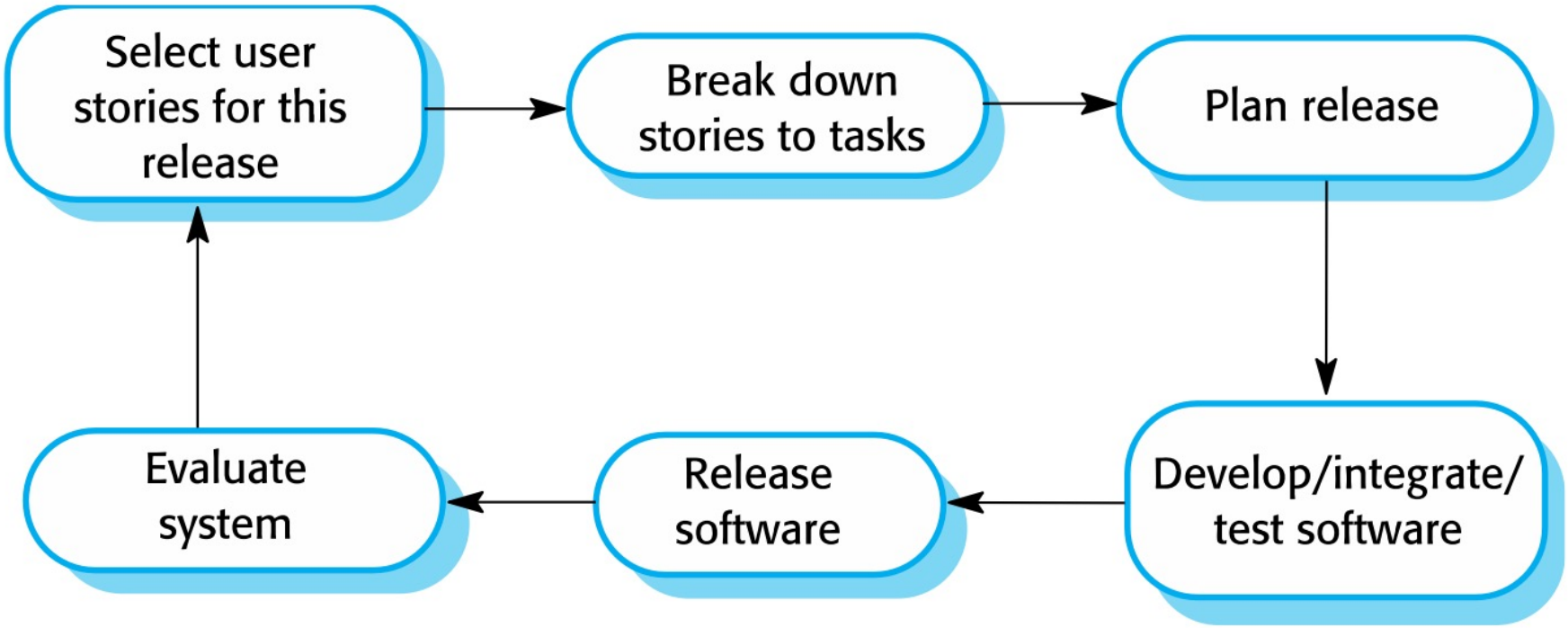
- Extreme Programming (“XP”)
- Scrum
- Kanban
- Feature-Driven Development
- Lean Development

EXTREAM PROGRAMMING

Extreme Programming

- A very influential agile method, developed in the late 1990s, that introduced a range of agile development techniques.
- Extreme Programming (XP) takes an ‘extreme’ approach to iterative development.
  - New versions may be built several times per day;
  - Increments are delivered to customers every 2 weeks;
  - All tests must be run for every build and the build is only accepted if tests run successfully

Extreme Programming - Release Cycle



Extreme Programming Practices

Principle or Practice	Description
Incremental planning	Requirements are recorded on story cards and the stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development 'Tasks'.
Small releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Simple design	Enough design is carried out to meet the current requirements and no more.
TDD	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.

Extreme Programming Practices

Principle or Practice	Description
Pair programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
Collective ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all of the code. Anyone can change anything.
Continuous integration	As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium term productivity
On-site customer	A representative of the end-user of the system (the customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

Influential XP practices

- Extreme programming has a technical focus and is not easy to integrate with management practice in most organizations.
- Agile development uses practices from XP
- Key practices
  - User stories for specification
  - Refactoring
  - Test-first development
  - Pair programming

User Stories

- In XP, a customer or user is part of the XP team and is responsible for making decisions on requirements.
- User requirements are expressed as user stories or scenarios.
- These are written on cards and the development team break them down into implementation tasks. These tasks are the basis of schedule and cost estimates.
- The customer chooses the stories for inclusion in the next release based on their priorities and the schedule estimates

User Story

Prescribing medication

The record of the patient must be open for input. Click on the medication field and select either 'current medication', 'new medication' or 'formulary'.

If you select 'current medication', you will be asked to check the dose; If you wish to change the dose, enter the new dose then confirm the prescription.

If you choose, 'new medication', the system assumes that you know which medication you wish to prescribe. Type the first few letters of the drug name. You will then see a list of possible drugs starting with these letters. Choose the required medication. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

If you choose 'formulary', you will be presented with a search box for the approved formulary. Search for the drug required then select it. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

In all cases, the system will check that the dose is within the approved range and will ask you to change it if it is outside the range of recommended doses.

After you have confirmed the prescription, it will be displayed for checking. Either click 'OK' or 'Change'. If you click 'OK', your prescription will be recorded on the audit database. If you click 'Change', you reenter the 'Prescribing medication' process.

Tasks

Task 1: Change dose of prescribed drug

Task 2: Formulary selection

Task 3: Dose checking

Dose checking is a safety precaution to check that the doctor has not prescribed a dangerously small or large dose.
Using the formulary id for the generic drug name, lookup the formulary and retrieve the recommended maximum and minimum dose.
Check the prescribed dose against the minimum and maximum. If outside the range, issue an error message saying that the dose is too high or too low.
If within the range, enable the 'Confirm' button.

Refactoring

- Conventional wisdom in software engineering is to design for change. It is worth spending time and effort anticipating changes as this reduces costs later in the life cycle.
- XP, however, maintains that this is not worthwhile as changes cannot be reliably anticipated.
- Rather, it proposes **constant code improvement** (refactoring) to make changes easier when they have to be implemented

Refactoring

- Programming team look for possible software improvements and make these improvements even where there is no immediate need for them.
- This improves the understandability of the software and so reduces the need for documentation.
- Changes are easier to make because the code is well structured and clear.
- However, some changes requires architecture refactoring and this is much more expensive.

Examples

- Re-organization of a class hierarchy to remove duplicate code.
- Tidying up and renaming attributes and methods to make them easier to understand.
- The replacement of inline code with calls to methods that have been included in a program library.

Test-first (driven) Development (TDD)

- Testing is central to XP and XP has developed an approach where the program is tested after every change has been made.
- XP testing features:
  - Test-first development.
  - Incremental test development from scenarios.
  - User involvement in test development and validation.
  - Automated test harnesses are used to run all component tests each time that a new release is built.

Test-first (driven) Development (TDD)

- Writing tests before code clarifies the requirements to be implemented.
- Tests are written as programs rather than data so that they can be executed automatically. The test includes a check that it has executed correctly.
  - Usually relies on a testing framework such as Junit.
- All previous and new tests are run automatically when new functionality is added, thus checking that the new functionality has not introduced errors.

Example of a Test Case

Test 4: Dose checking

Input:
1. A number in mg representing a single dose of the drug.
2. A number representing the number of single doses per day.

Tests:
1. Test for inputs where the single dose is correct but the frequency is too high.
2. Test for inputs where the single dose is too high and too low.
3. Test for inputs where the single dose * frequency is too high and too low.
4. Test for inputs where single dose * frequency is in the permitted range.

Output:
OK or error message indicating that the dose is outside the safe range.

Pair Programming

- Pair programming involves programmers working in pairs, developing code together.
- This helps develop common ownership of code and spreads knowledge across the team.
- It serves as an informal review process as each line of code is looked at by more than 1 person.
- It encourages refactoring as the whole team can benefit from improving the system code.

Pair Programming

- In pair programming, programmers sit together at the same computer to develop the software.
- Pairs are created dynamically so that all team members work with each other during the development process.
- The sharing of knowledge that happens during pair programming is very important as it reduces the overall risks to a project when team members leave.
- Pair programming is not necessarily inefficient and there is some evidence that suggests that a pair working together is more efficient than 2 programmers working separately.

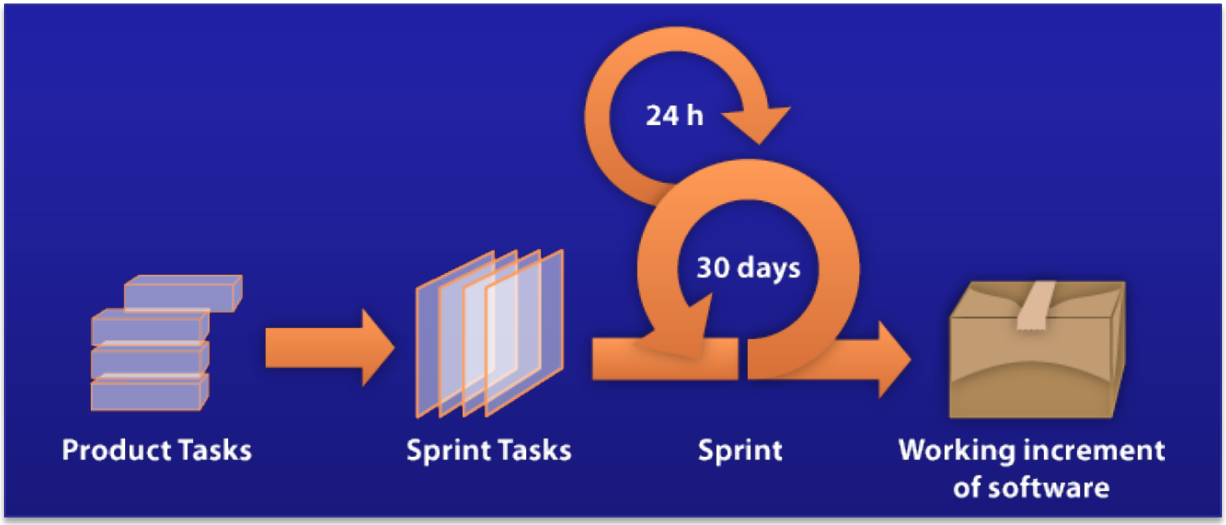
SCRUM

What is Scrum?

- **Scrum** is a software engineering development framework within which people can address complex problems, while productively and creatively delivering products of the highest possible value.
- Scrum was developed in the early 1990s by Ken Schwaber and Jeff Sutherland
- Scrum is:
  - Lightweight
  - Simple to understand
  - Easy to use

Who uses Scrum?

Scrum Framework



Roles

- Product Owner
- Scrum Master
- Development Team

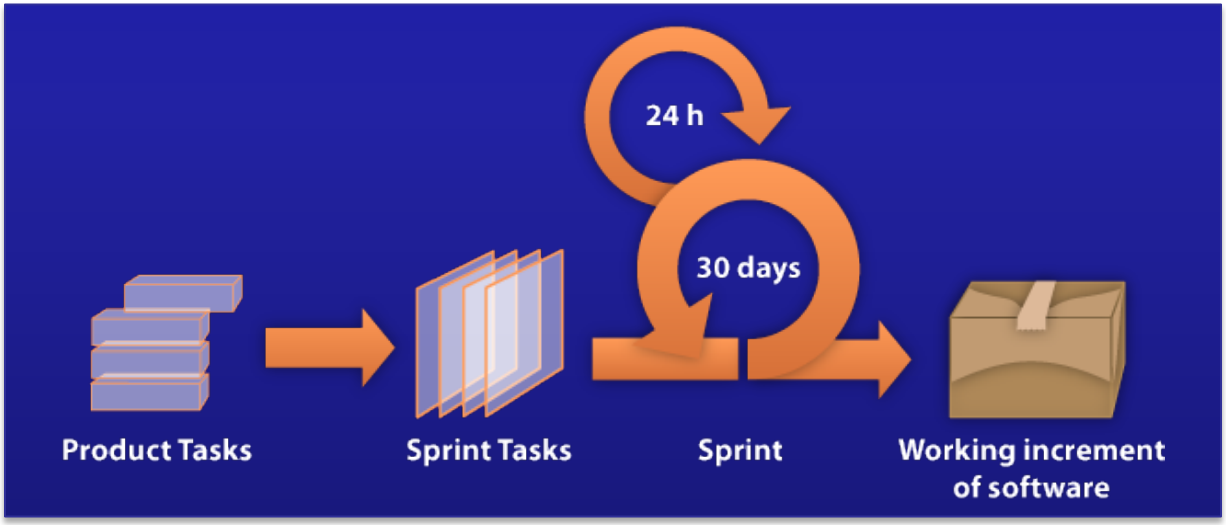
Events

- Sprint
- Sprint Planning
- Daily Scrum
- Sprint Review
- Sprint Retrospective

Artifacts

- Project Charter
- Product Backlog
- Sprint Backlog
- Burn down chart

Scrum Framework



Roles

- Product Owner
- Scrum Master
- Development Team

Events

- Sprint
- Sprint Planning
- Daily Scrum
- Sprint Review
- Sprint Retrospective

Artifacts

- Project Charter
- Product Backlog
- Sprint Backlog
- Burn down chart

Product Owner

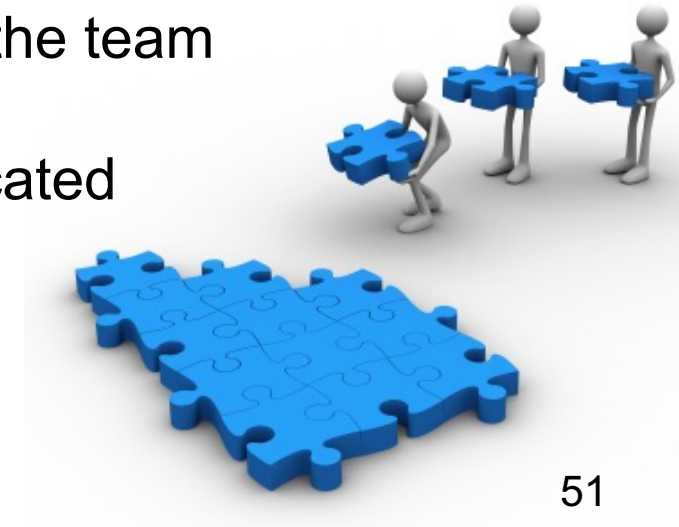
- TEAM
  - Defines the features of the product
  - Decides on release date and content
  - Prioritizes features according to market value
  - Adjusts features and priority every iteration, as needed
- PROJECT COORDINATOR
  - Accepts or rejects work results

Scrum Master

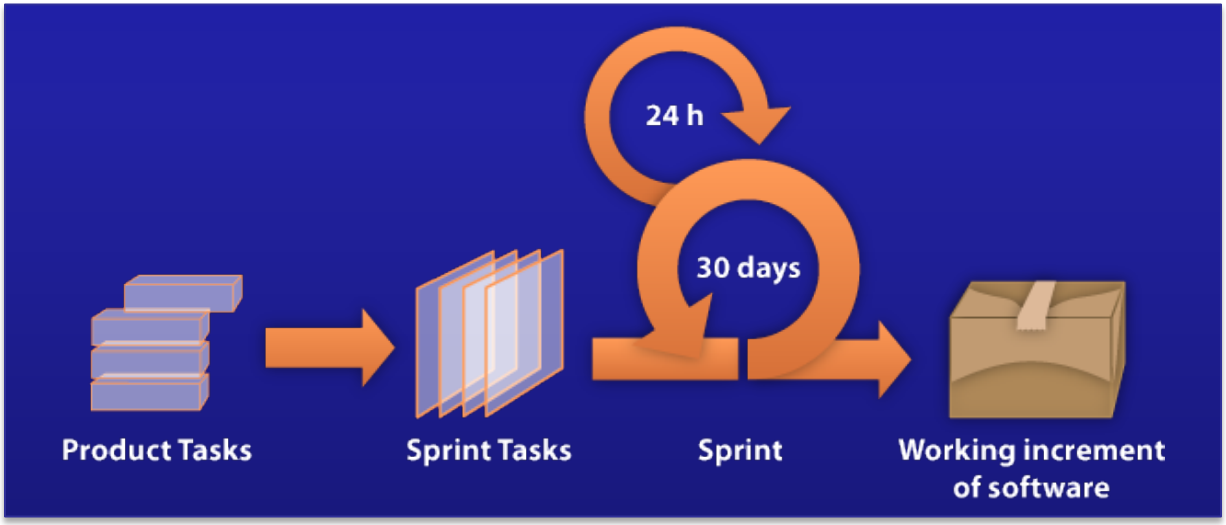
- TEAM LEADER
  - Represents management to the project
  - Responsible for enacting Scrum values and practices
  - Removes impediments
  - Ensures that the team is fully functional
  - Enables close cooperation across all roles and functions
  - Shields the team from external interferences

Development Team

- Members are cross-functional, with all of the skills as a team necessary to create a product
  - Programmers, user interface designers, testers, business analysts, etc.
- Members have no titles other than “Developer”, regardless of the work being performed by the person
- Members may have specialized skills and areas of focus, but accountability belongs to the team as a whole
- Teams do not contain sub-teams dedicated to particular domains like testing or business analysis.
- Optimal team size: 4-6 members



Scrum Framework



Roles

- Product Owner
- Scrum Master
- Development Team

Events

- Sprint
- Sprint Planning
- Daily Scrum
- Sprint Review
- Sprint Retrospective

Artifacts

- Product Backlog
- Sprint Backlog

Sprint

- Considered the “heart” of Scrum
- A **sprint** is a time-box of one month or less (usually between 2-4 weeks) during which a “Done,” useable, and potentially-releasable product increment is created.
- Sprints have consistent durations throughout a development effort.
- A new Sprint starts immediately after the conclusion of the previous Sprint.
- During the Sprint:
  - No changes are made that will affect the Sprint Goal
  - Development Team composition remains constant
  - Software Quality attributes do not change
  - Scope may be clarified and re-negotiated between the Product Owner and Development Team as more is learned

Sprint Planning Meeting

- A **Sprint Planning Meeting** is a meeting where the work to be performed in the sprint is planned.
- Usually time-boxed to eight hours for a one-month sprint. For shorter sprints, the event is proportionately shorter.
- There are two parts to the meeting:
  - What will be delivered in this Sprint?
  - How will the work get done?

What will be delivered in this sprint?

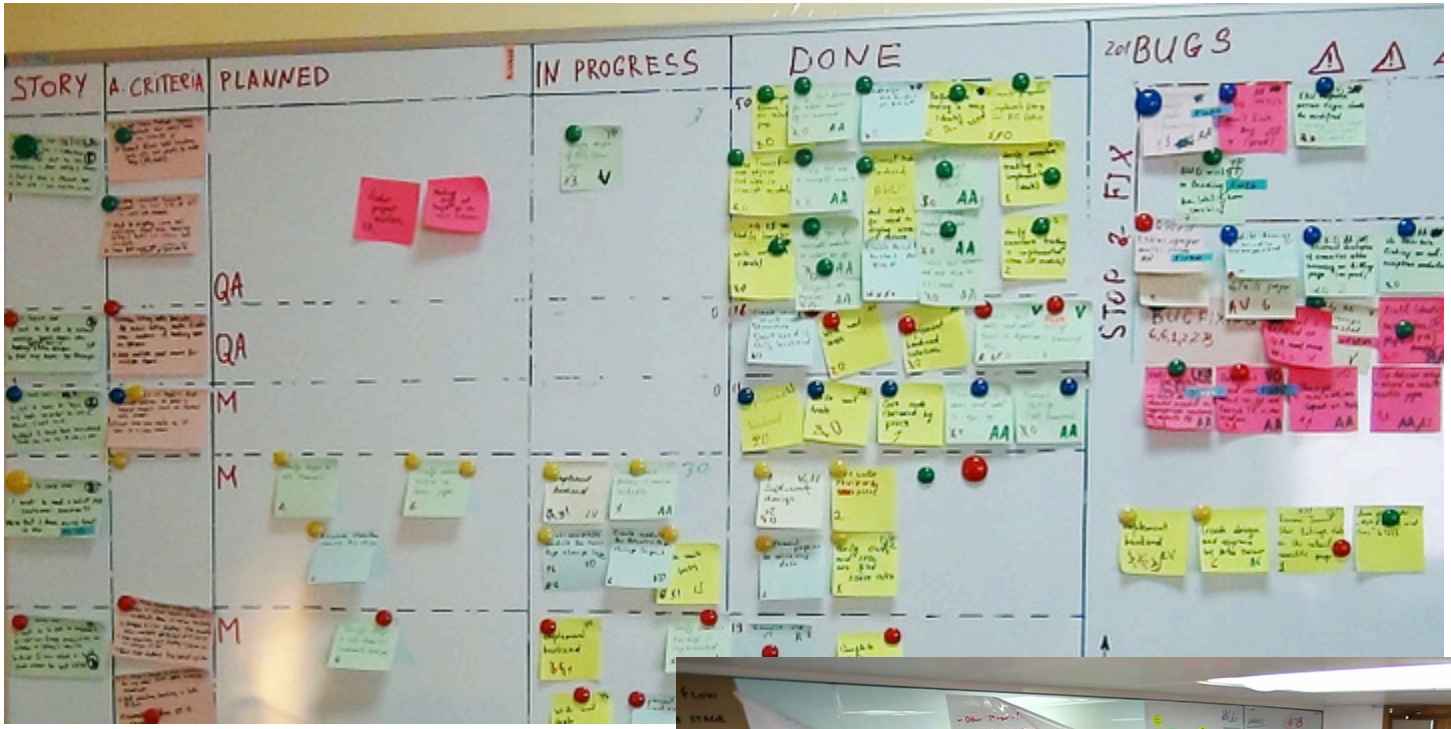
- The Product Owner presents to the Development Team an ordered list of all items needed to be completed.
- The entire Scrum Team collaborates on understanding the work to be done in the Sprint.
- Other items evaluated before making a decision:
  - Product Backlog
  - The latest product Increment (or Iteration)
  - Projected capacity of the Development Team for the upcoming Sprint
  - Past performance of the Development Team
- A **Sprint Goal**, or an objective that will be met within the Sprint, is defined during this meeting.

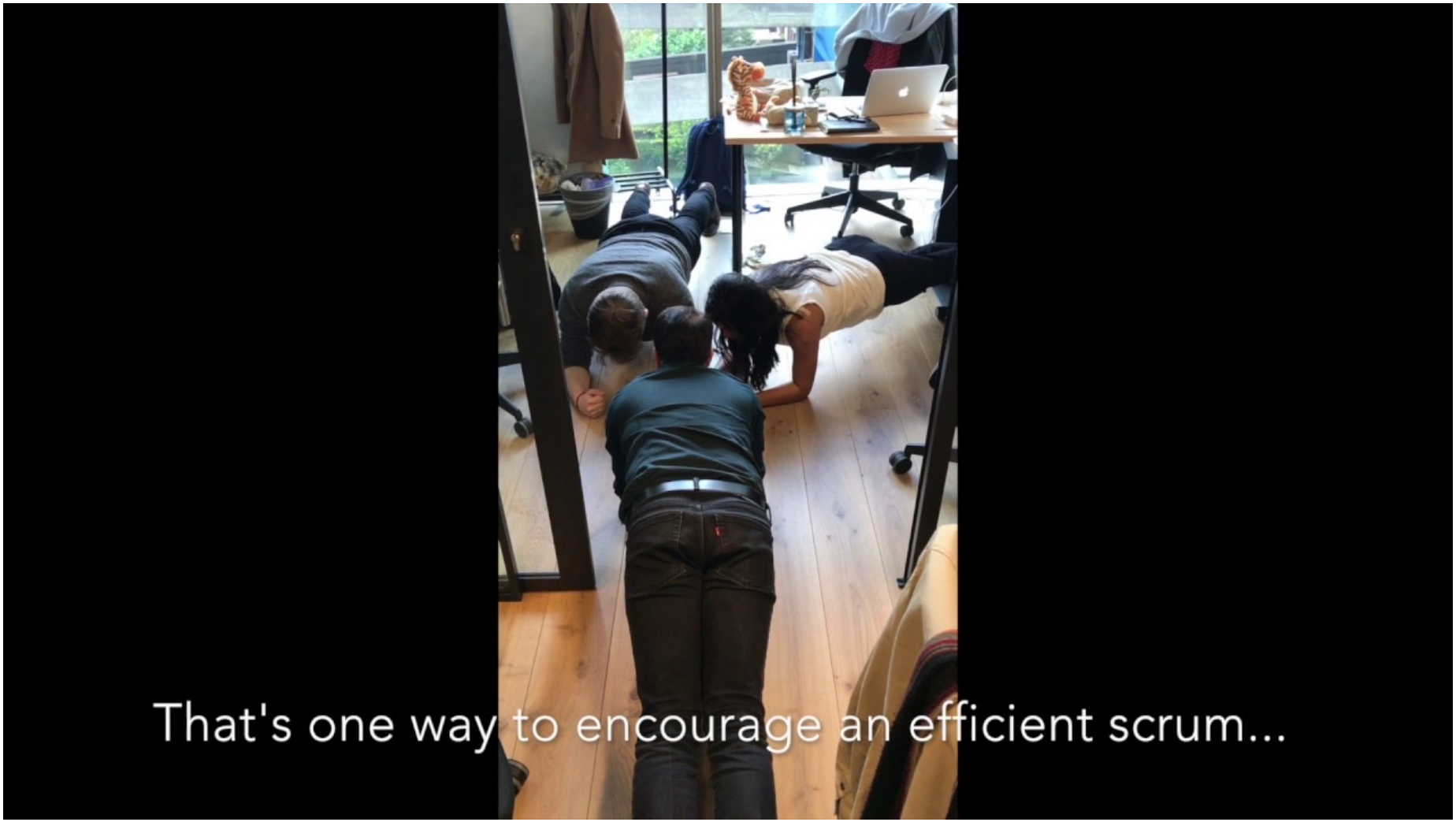
How will the work get done?

- The Development Team decides how it will build this functionality into a “Done” product Increment during the Sprint
- The **Product Backlog** items selected for this Sprint plus the plan for delivering them is called the **Sprint Backlog**.
- The Development Team self-organizes the **Sprint Backlog** by estimating the time it will take to complete each item and assigning tasks to individual team members

Daily Scrum Meeting

- The Daily Scrum (or “Stand-up Meeting”) is a 15-minute time-boxed event for the Development Team to synchronize activities and create a plan for the next 24 hours.
- The Daily Scrum is held at the same time and place each day to reduce complexity.
- Each team member answers three questions:
  - What has been accomplished since the last meeting?
  - What will be done before the next meeting?
  - What obstacles are in the way?


STORY
A. CRITERIA
PLANNED
IN PROGRESS
DONE
201 BUGS
STOP & FIX
QA
QA
M
M
M


That's one way to encourage an efficient scrum...

Daily Scrum Advantages

- Improves communication among team members
- Eliminates the need for other meetings
- Identifies and removes impediments to development
- Highlights and promotes quick decision-making
- Improves the Development Team’s level of project knowledge.

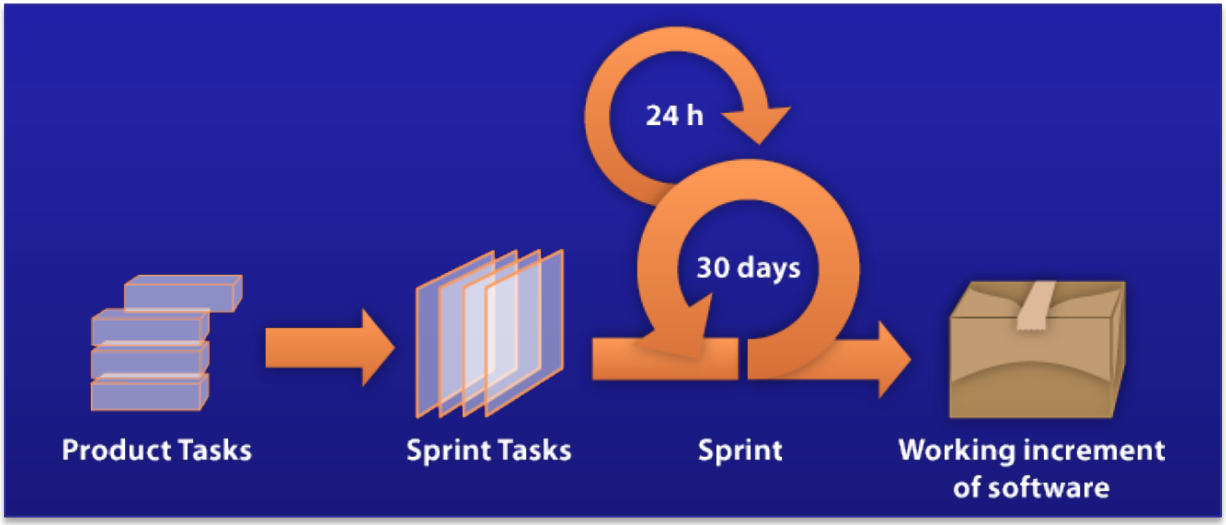
Sprint Review Meeting

- The Sprint Review Meeting is the time when the Development Team presents what it accomplished during the sprint
- Takes the form of a ***demo*** of new features or concrete progress
- Informal and requires little prep time
- The entire team participates

Sprint Retrospective Meeting

- The Sprint Retrospective is an opportunity for the Scrum Team to reflect and create a plan for improvements to be enacted during the next Sprint.
- The purpose:
  - Think about how the last Sprint went with regards to people, relationships, process, and tools
  - Identify the major items that went well and potential improvements
  - Create a plan for implementing improvements to the way the Scrum Team does its work
- All team members have the opportunity to answer:
  - What went well during the last sprint?
  - What didn’t go well during the last sprint?
  - How should the team improve for the next sprint?

Scrum Framework



Roles

- Product Owner
- Scrum Master
- Development Team

Events

- Sprint
- Sprint Planning
- Daily Scrum
- Sprint Review
- Sprint Retrospective

Artifacts

- Project Charter
- Product Backlog
- Sprint Backlog
- Burn down chart

Project Charter

- Problem Statement: Short and succinct (one or two sentences)
- Project Objectives: What the project will achieve
- Stakeholders: Persons who will be actively involved with the project (e.g. project sponsor, types of users, etc.)
- Project Deliverables: The major results or services that will be produced, what are the specific things the software will do

Product Backlog

- The **Product Backlog** is an ordered list of everything that might be needed in the product and is the single source of **Requirements** for any changes to be made to the product.
- The Product Owner is responsible for the Product Backlog, including its content, availability, and ordering.
- A Product Backlog is never complete.
- Lists all features, functions, requirements, enhancements, and fixes that constitute the changes to be made to the product in future releases.

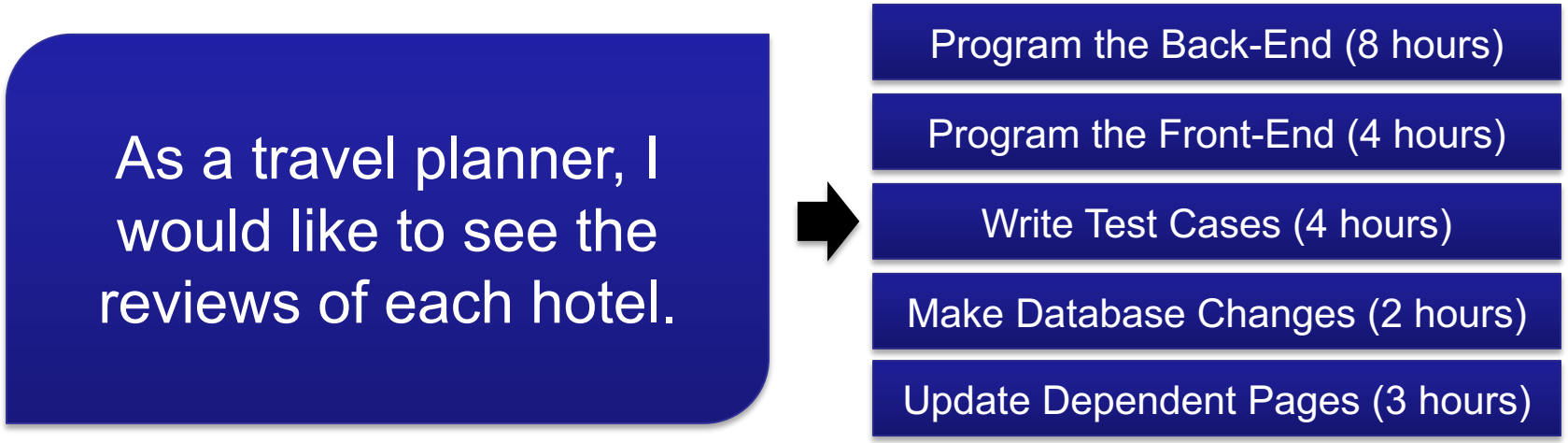
Product Backlog Sample

Backlog Item
As a guest, I want to make a reservation.
As a guest, I want to cancel a reservation.
As a guest, I want to change the dates of a reservation.
As an admin, I want to change the availability of dates at my hotel.
As a developer, I want to improve exception handling.

- Backlog items are usually in the form of:
  - As a _____, I want to _____ (so that I can _____).
- Product Backlog items are sometimes called “user stories”

Sprint Backlog

- In a **Sprint Backlog**, the Development Team selects items from the product backlog they can commit to completing in a given sprint.
- The team then identifies tasks to complete these and each is estimated (how many hours to complete).



User Story

Sprint Backlog Items

Managing the Sprint Backlog

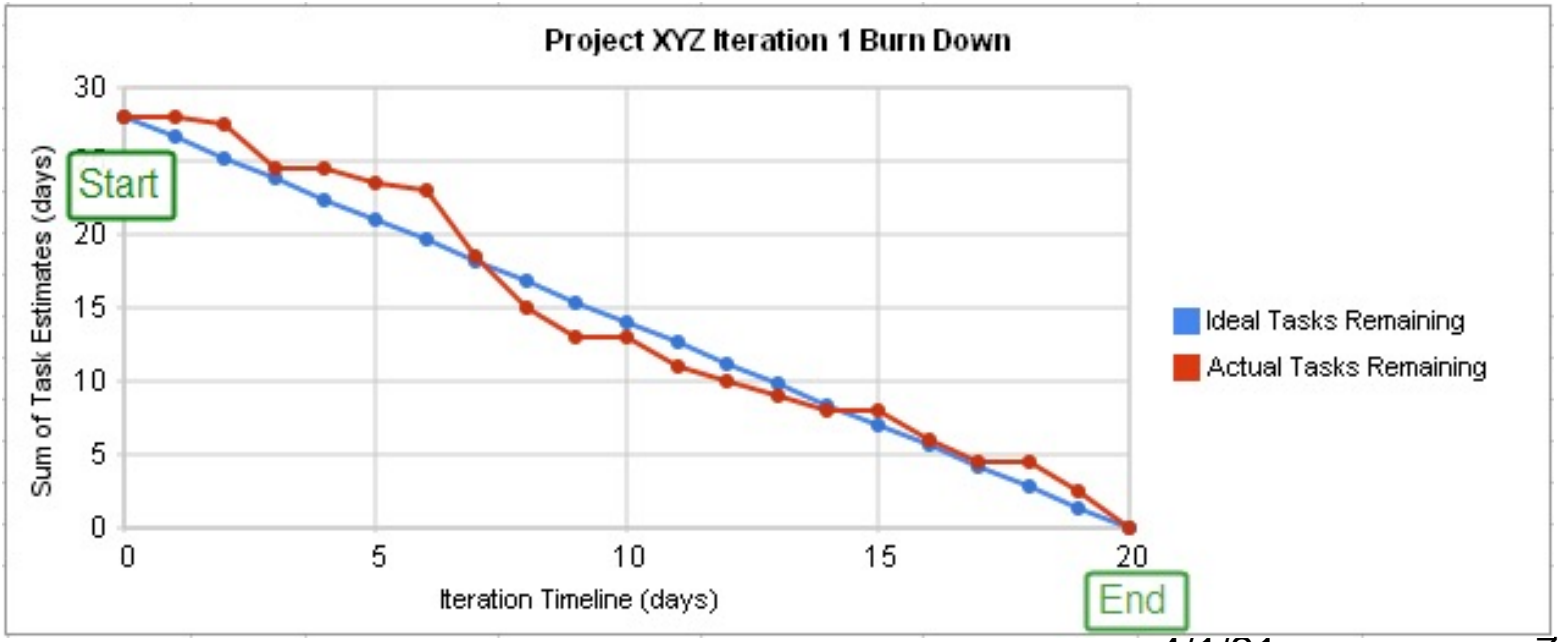
- Individuals sign up for work of their own choosing
  - Work is never assigned
- Estimated work remaining is updated daily
- Any team member can add to, delete from, or modify the sprint backlog
- Work for the sprint emerges
- If work is unclear, define a sprint backlog item with a larger amount of time and break it down later
- Update work remaining as more becomes known

Sprint Backlog Sample

Backlog Tasks	Mon	Tue	Wed	Thur	Fri
Program the Back-End	3	4	1		
Program the Front-End			2	2	
Write Test Cases	3		1		
Make Database Changes		2			
Update Dependent Pages					3

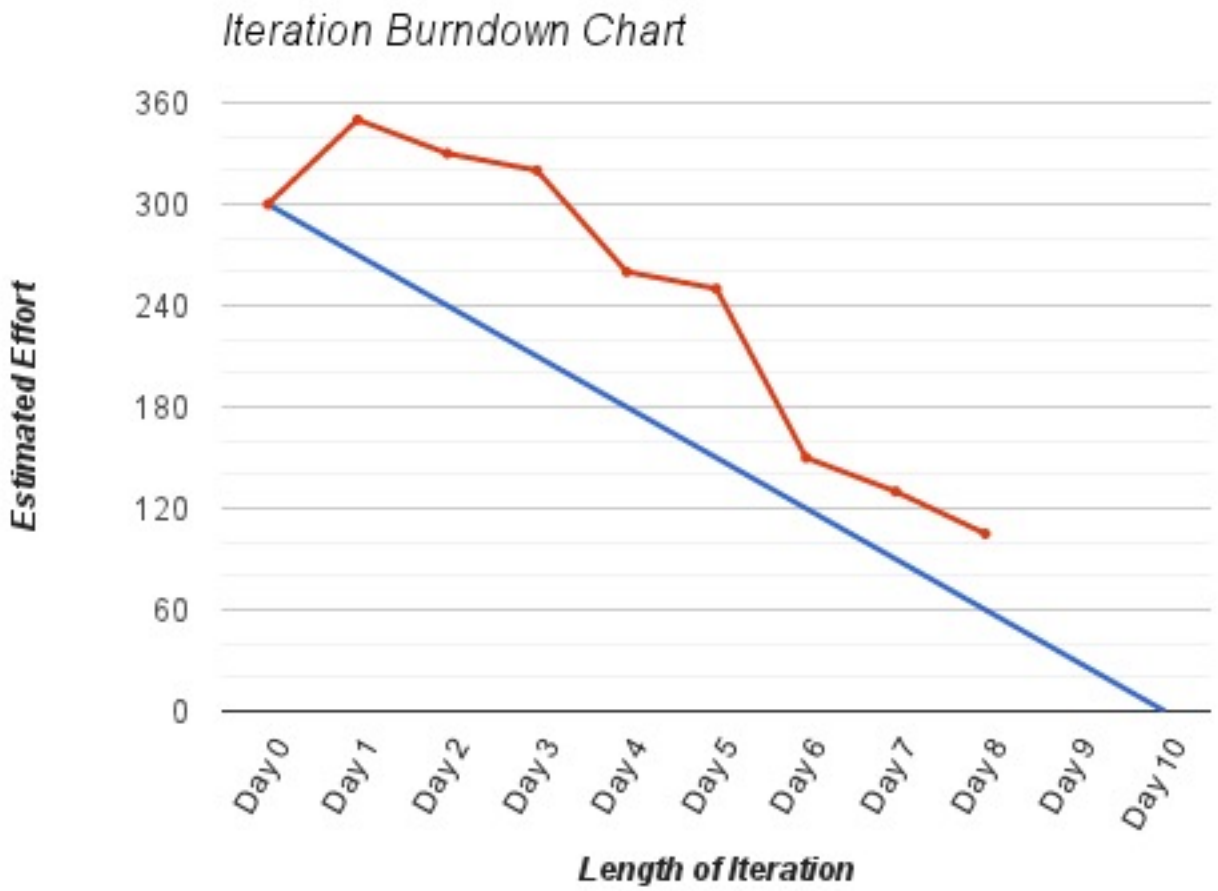
Burn Down Chart

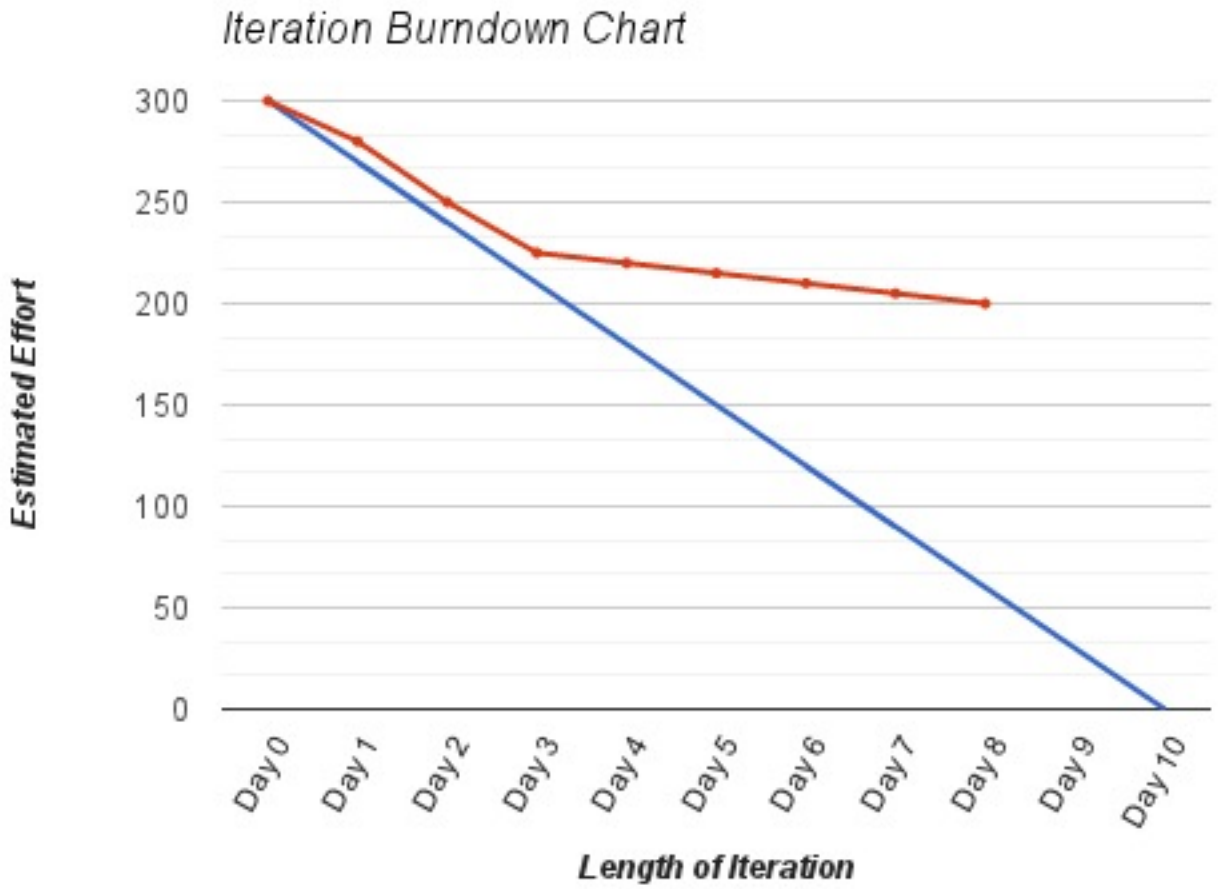
- Visualize the correlation between the amount of work remaining and the progress in reducing the work
  - X: date
  - Y: hours of work remaining
- Updated according the Sprint backlog

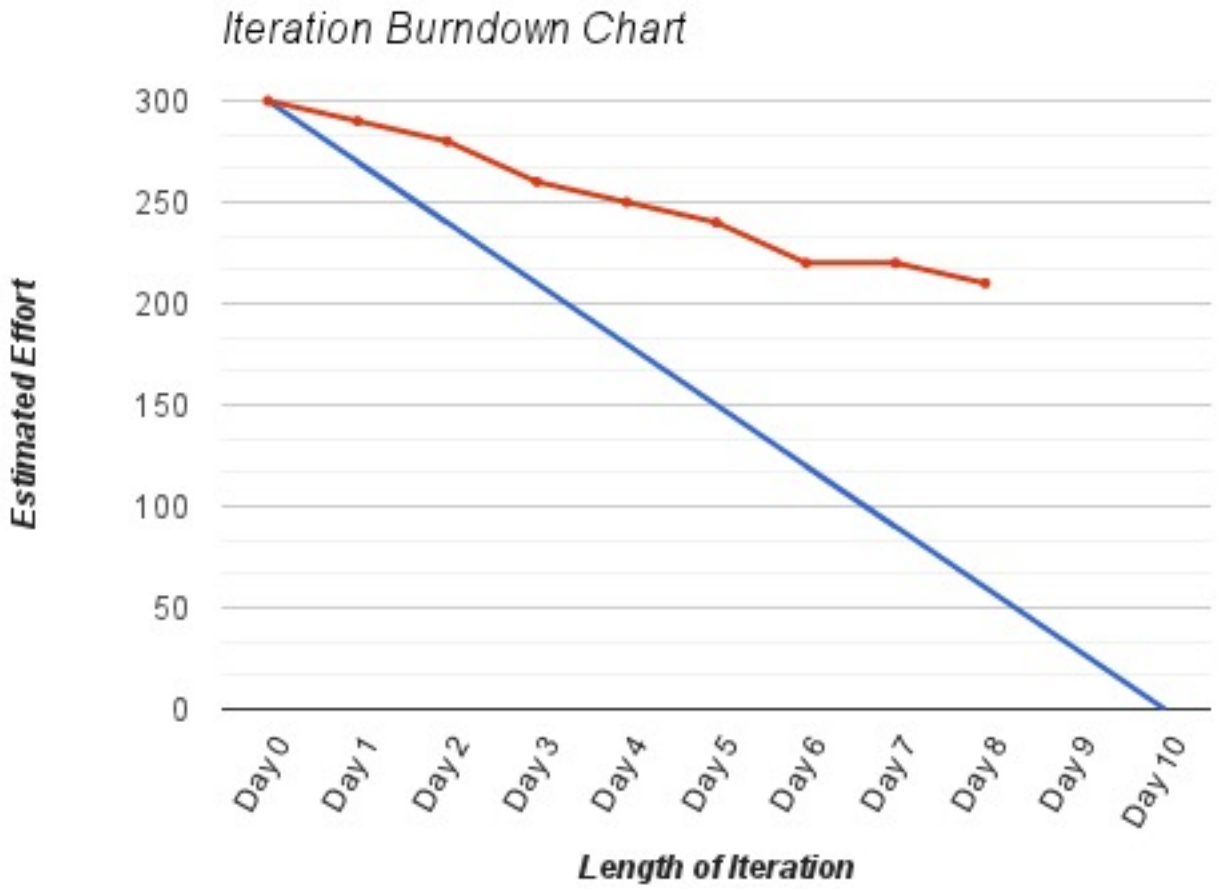


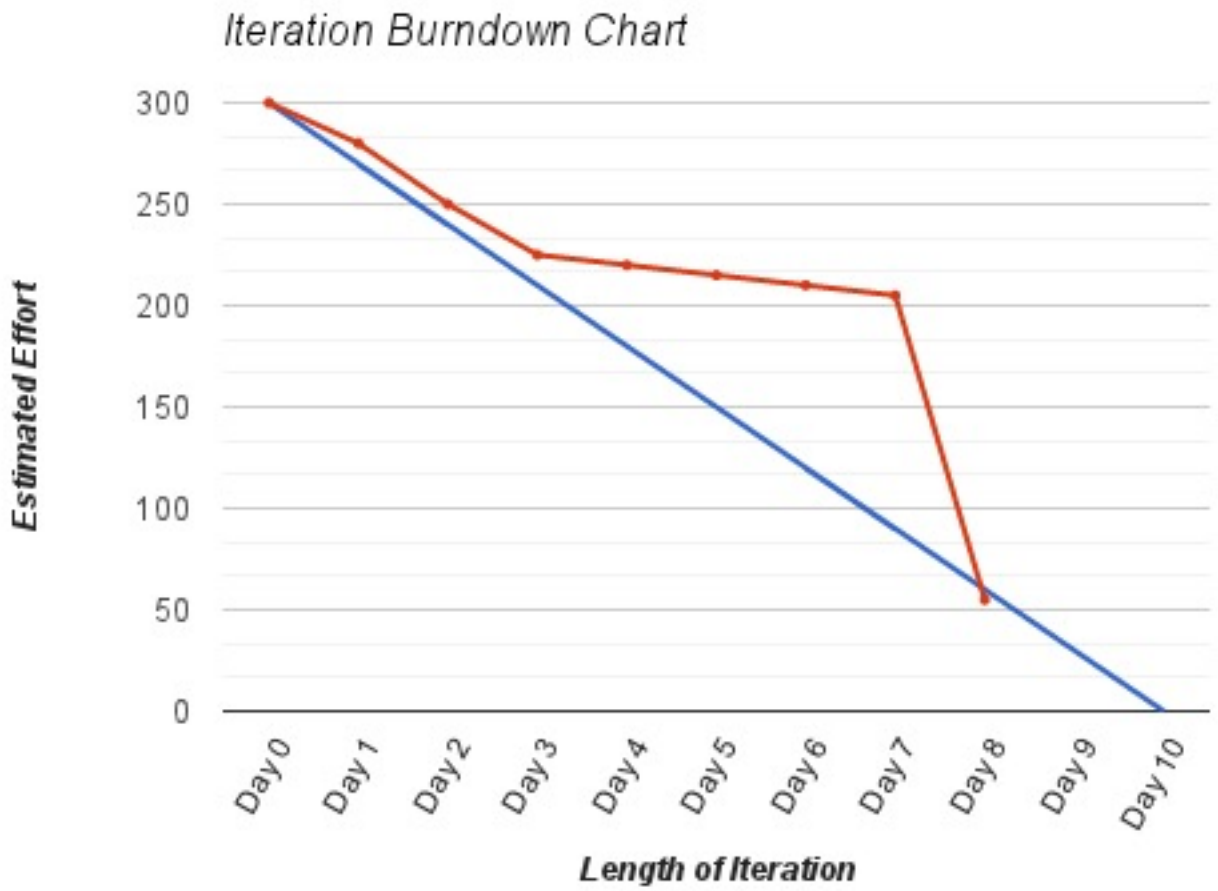
Problems With Burn Down Charts

- Burn down charts have a major assumption
  - **The amount of work does not change**
    - Actually the amount of work in the project is changing
      - More work is added
      - At the end of the project, a large amount of work was removed to make the deadline

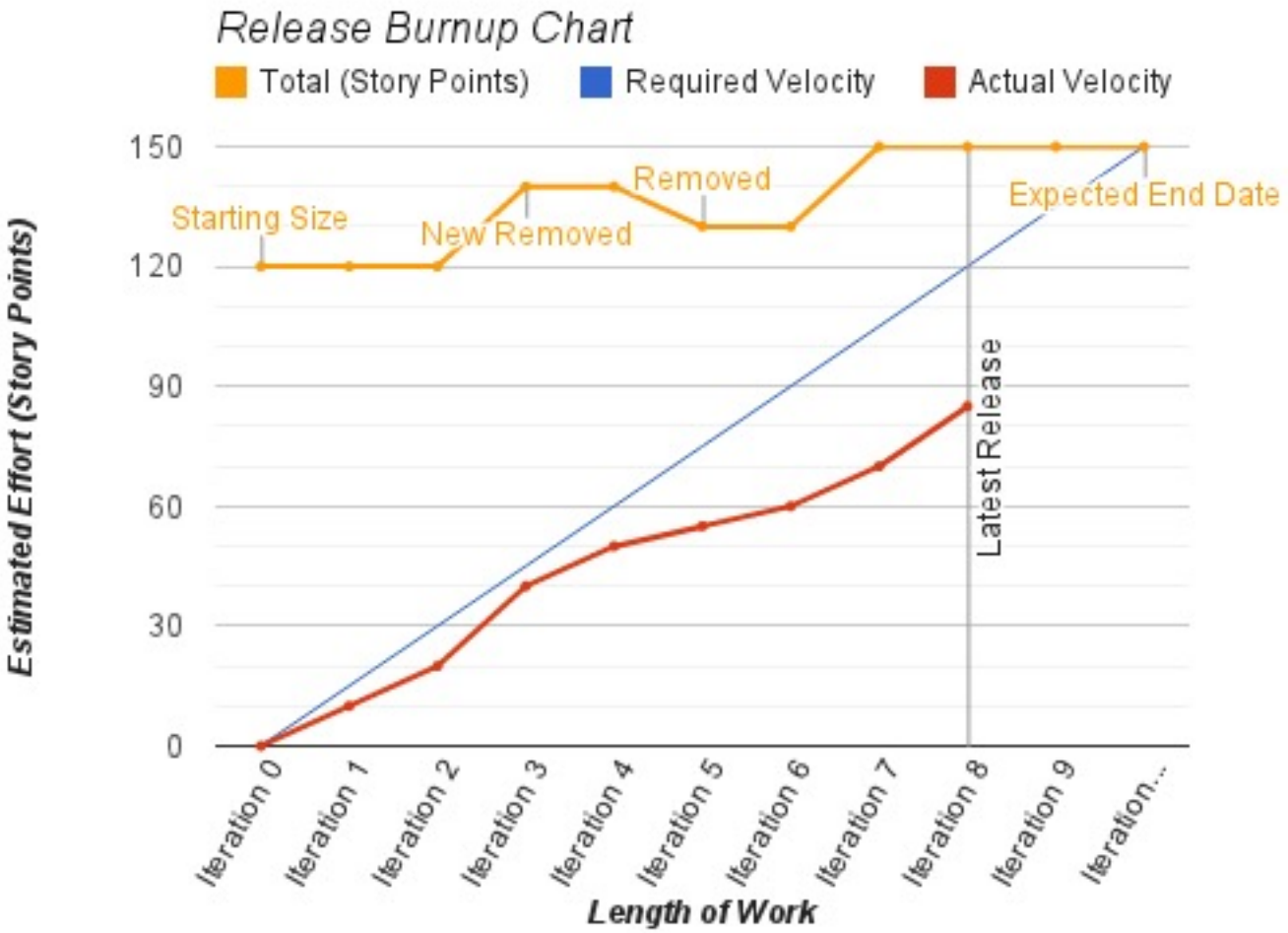

Iteration Burndown Chart
360
300
240
180
120
60
0
Estimated Effort
Day 0
Day 1
Day 2
Day 3
Day 4
Day 5
Day 6
Day 7
Day 8
Day 9
Day 10
Length of Iteration


Iteration Burndown Chart
300
250
200
150
100
50
0
Estimated Effort
Day 0
Day 1
Day 2
Day 3
Day 4
Day 5
Day 6
Day 7
Day 8
Day 9
Day 10
Length of Iteration


Iteration Burndown Chart
300
250
200
150
100
50
0
Estimated Effort
Day 0
Day 1
Day 2
Day 3
Day 4
Day 5
Day 6
Day 7
Day 8
Day 9
Day 10
Length of Iteration

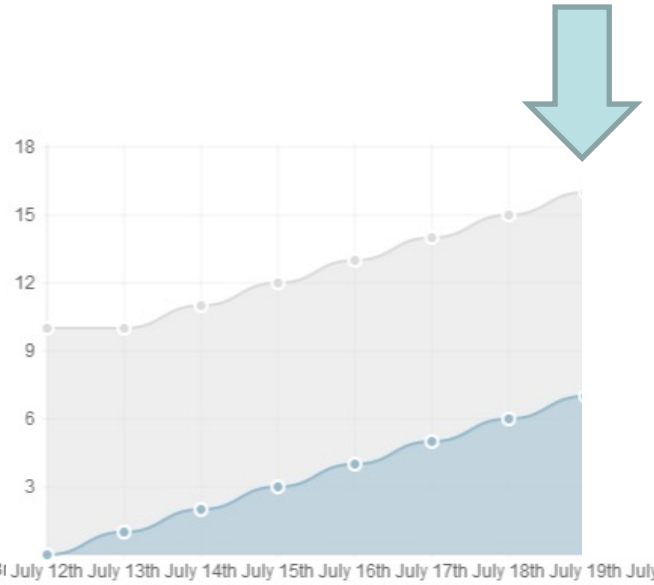

Iteration Burndown Chart
300
250
200
150
100
50
0
Estimated Effort
Day 0
Day 1
Day 2
Day 3
Day 4
Day 5
Day 6
Day 7
Day 8
Day 9
Day 10
Length of Iteration

Burnup Chart



Burn Down vs. Burn Up

- In the burn down chart it looks like the team is not making any progress. In the burn up chart it is clear the team is making progress but new work is being added as fast as they can complete it.



Burn Down vs. Burn Up

- In the burn down chart it looks like the team is made a heroic effort at the end. In the burn up chart it is clear that work was removed from the project to meet the deadline. (The work may have been moved to the next version or milestone)