Agile Development Processes

CSCE 740 - Lecture 3 - 08/31/2017

Common Practice: Code & Fix

Sit down, write out the code, and fix problems as they occur. No formal structure to development.

What is wrong with this scenario?

- Often results in a loop of bug fixes that introduce new bugs.
  - Ties up developers.
- Hard to track development progress.
- **Introduces unnecessary risk.**

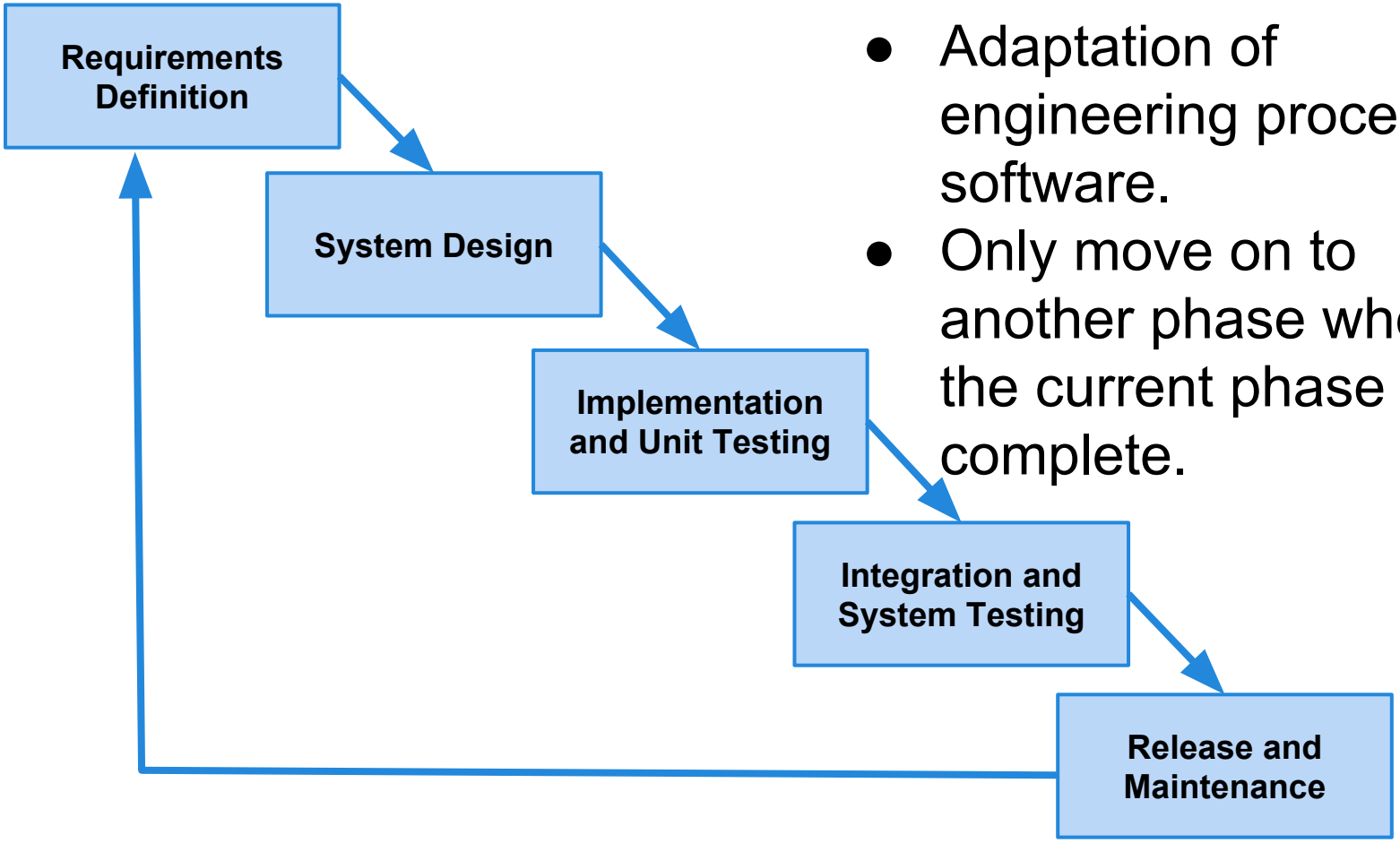
We Need A Process

A formal process structures the development of the software into a series of visible phases.

As a result:

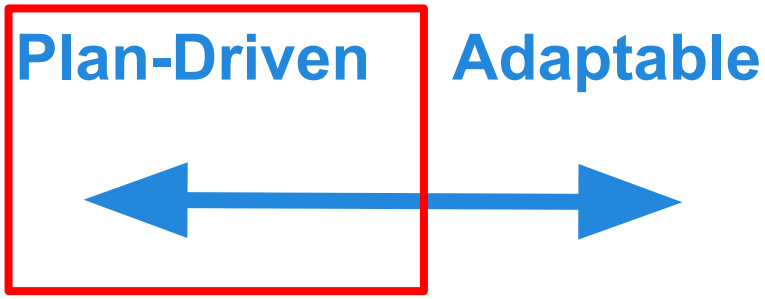
- We have control over the project.
  - Visibility into development progress.
  - Knowledge of when to move forward.
- Developers are more efficient.
- Risks can be anticipated and mitigated.

The Waterfall Model



- Adaptation of engineering process to software.
- Only move on to another phase when the current phase is complete.

The Waterfall Model



Benefits of Waterfall?

- Spending more time on earlier phases prevents problems from being discovered later.
- Brings discipline and structure.
- Clear understanding of project progress.
- Places emphasis on documentation.

From The “Creator” Of Waterfall...

“I believe in the concept, but the implementation is risky and invites failure.”

- Winston Royce

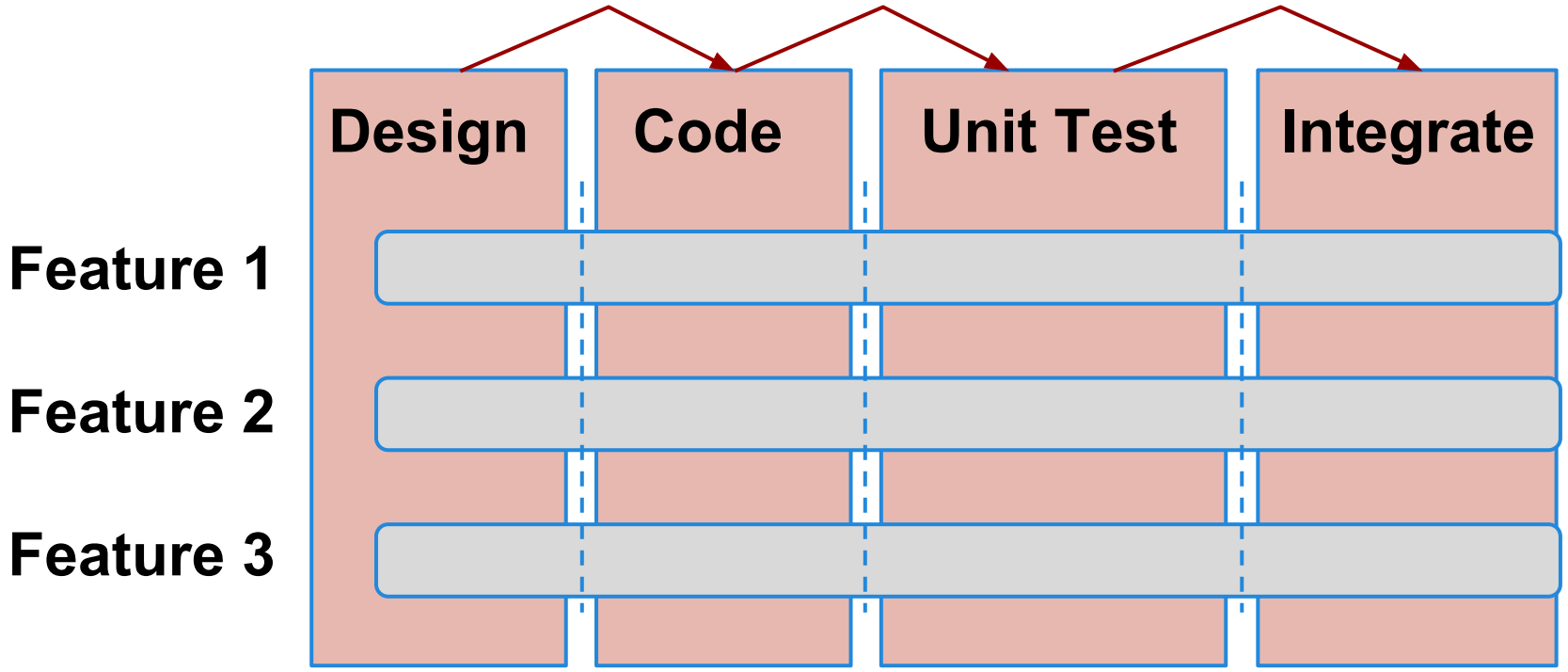
Why is the waterfall model risky?

- Inflexible model that **does not accommodate change.**
  - Hard to respond to unexpected risk.
- In practice, you need to return to earlier phases as details change.
  - You rarely know your requirements that early.
  - Implementation details often emerge only during implementation.

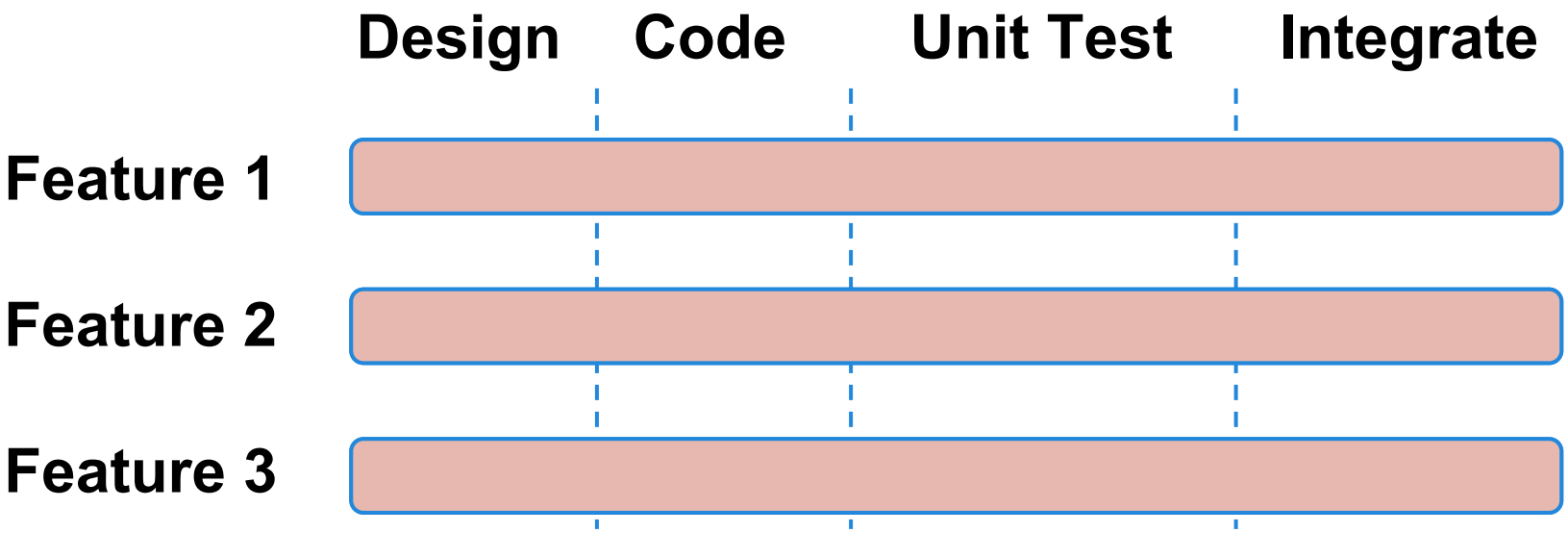
Enter… Agile Development

- Introduce “agile” software development processes
  - Iterative and Incremental Processes
  - The Agile Manifesto
  - The Scrum Process
- eXtreme Programming
  - Not a single process, but a set of agile principles that can guide development.

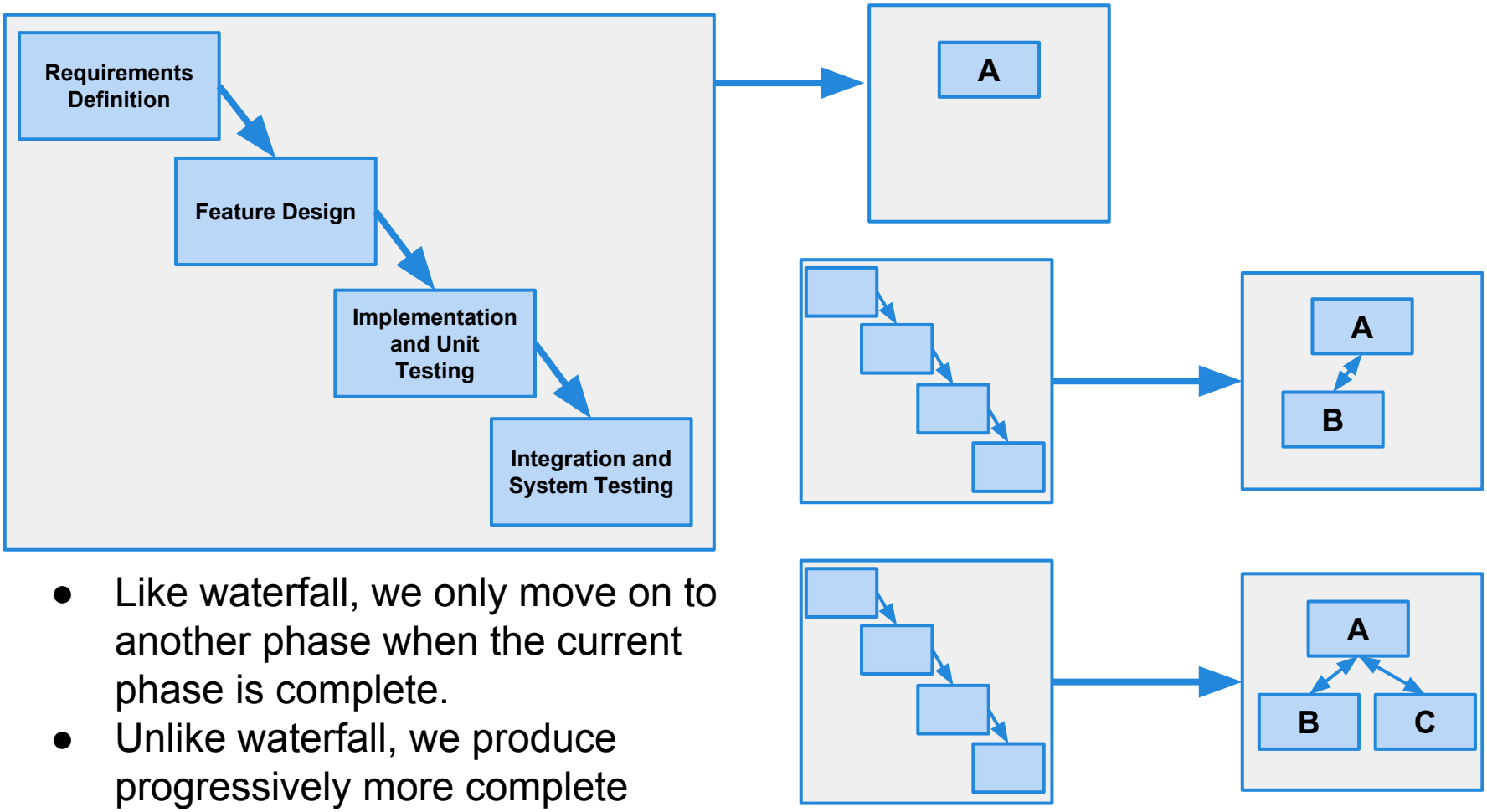
Work Partitioning: Waterfall



Work Partitioning: Incremental



The Incremental Model



- Like waterfall, we only move on to another phase when the current phase is complete.
- Unlike waterfall, we produce progressively more complete builds of a system.

The Incremental Model

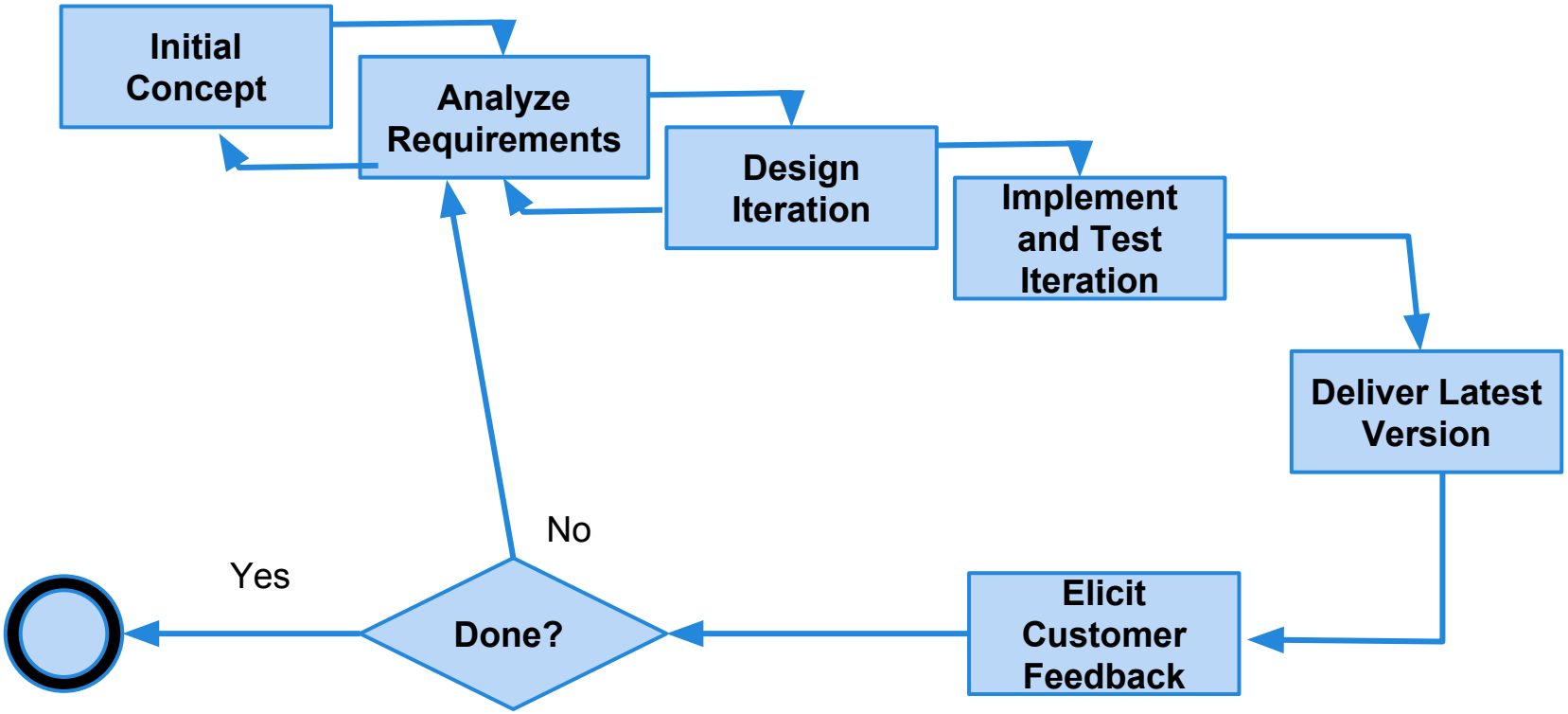
What are the advantages of incremental development?

- Customers can start using the system earlier.
  - and feedback can be obtained more frequently.
- Slower integration of features allows easier testing of individual features and feature interactions.
- If time/budget runs out, a partial product can still be released.

What are some of the disadvantages?

- Development is still rigid for each individual feature.
  - It is still hard to respond to feedback, as you may have to throw out all of the work for a particular feature.
- Still need to invest up-front planning for the “complete” system.

The Iterative/Evolutionary Model



Wait… Aren't incremental and iterative the same thing?

- **Incremental:** Add new features to build a progressively more complete system over time.
- **Iterative:** Deliver a series of progressively more complete prototypes over time.
- *Aren't these the same thing?*

Incremental: Writing an essay one "perfect" sentence at a time.

Iterative: Writing a complete rough draft, then improving it through a complete revision.

The Iterative Model

What are the advantages of iterative development?

- Frequent customer feedback can keep the project on track.
  - We throw away more work short-term, but far less long-term.
- Requirements and design can more easily be revised.
- Natural fit to how software is built - develop something “good enough” that can be revised over time.

What are some of the disadvantages?

- “Good enough” is very risky if there is software problems can result in harm to humans or their operating environment.
- Frequent releases often results in rushed releases.
  - Building on bad foundations results in a bad final product.

The Agile Manifesto

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

Agile Principles

- 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 with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.

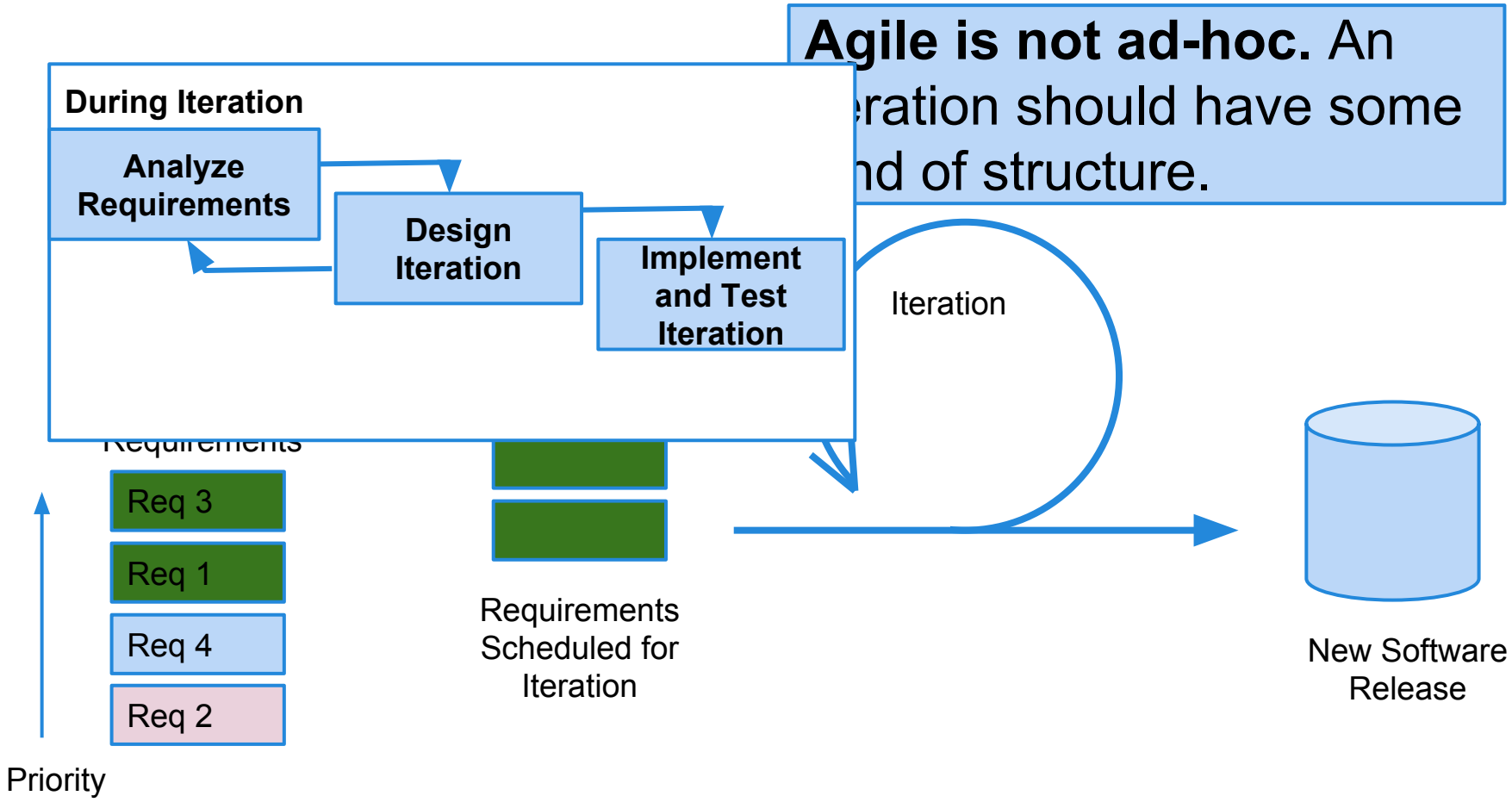
Agile Principles (2)

- 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.

Agile Principles (3)

- 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.

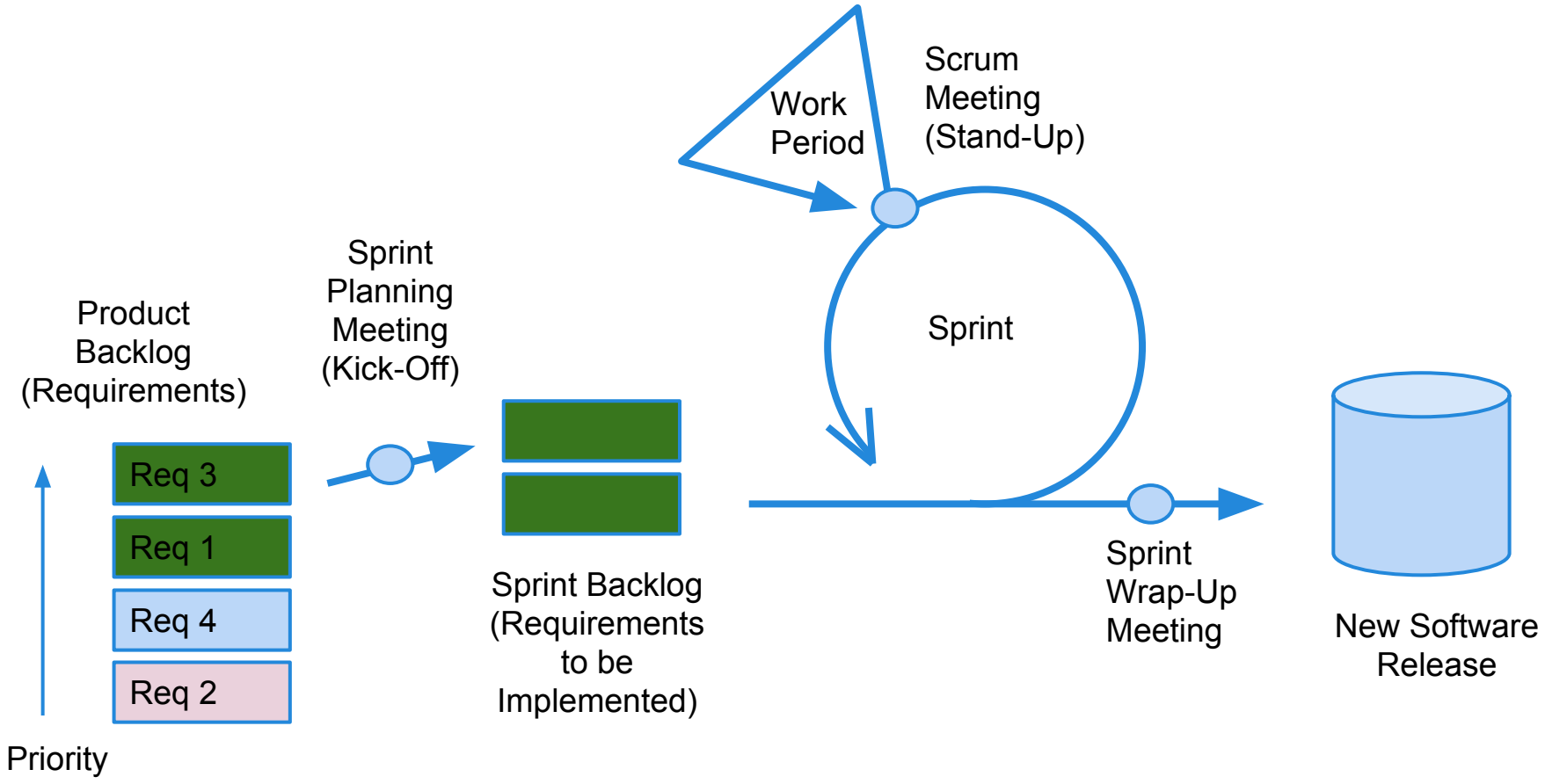
The Agile Model



The Scrum Model

- Bring the development team together to discuss problems as a team.
  - Then send them out to accomplish their individual goals.
- Individuals can define their own process.
  - But there is an overall structure, based on roles and meetings.

The Scrum Model



Scrum Roles

- **The Development Team**
  - Usually small, 3-9 people.
- **The Product Owner**
  - The “voice” of the customer.
  - Presents iterations to customers and communicates feedback to the team.
  - Maintains and prioritizes the requirement backlog
- **The Scrum Master**
  - The team “coach”.
  - Removing impediments to success.
  - Facilitating communication and meetings.
  - Mediating disputes.

Daily Scrum Meetings

- A daily 15-minute meeting in which all participants are standing.
- Each person answers three questions:
  - What did you complete since the last scrum?
  - What will you complete before the next scrum?
  - What, if any, blocking issues (impediments to progress) do you need to resolve?

The Scrum Model

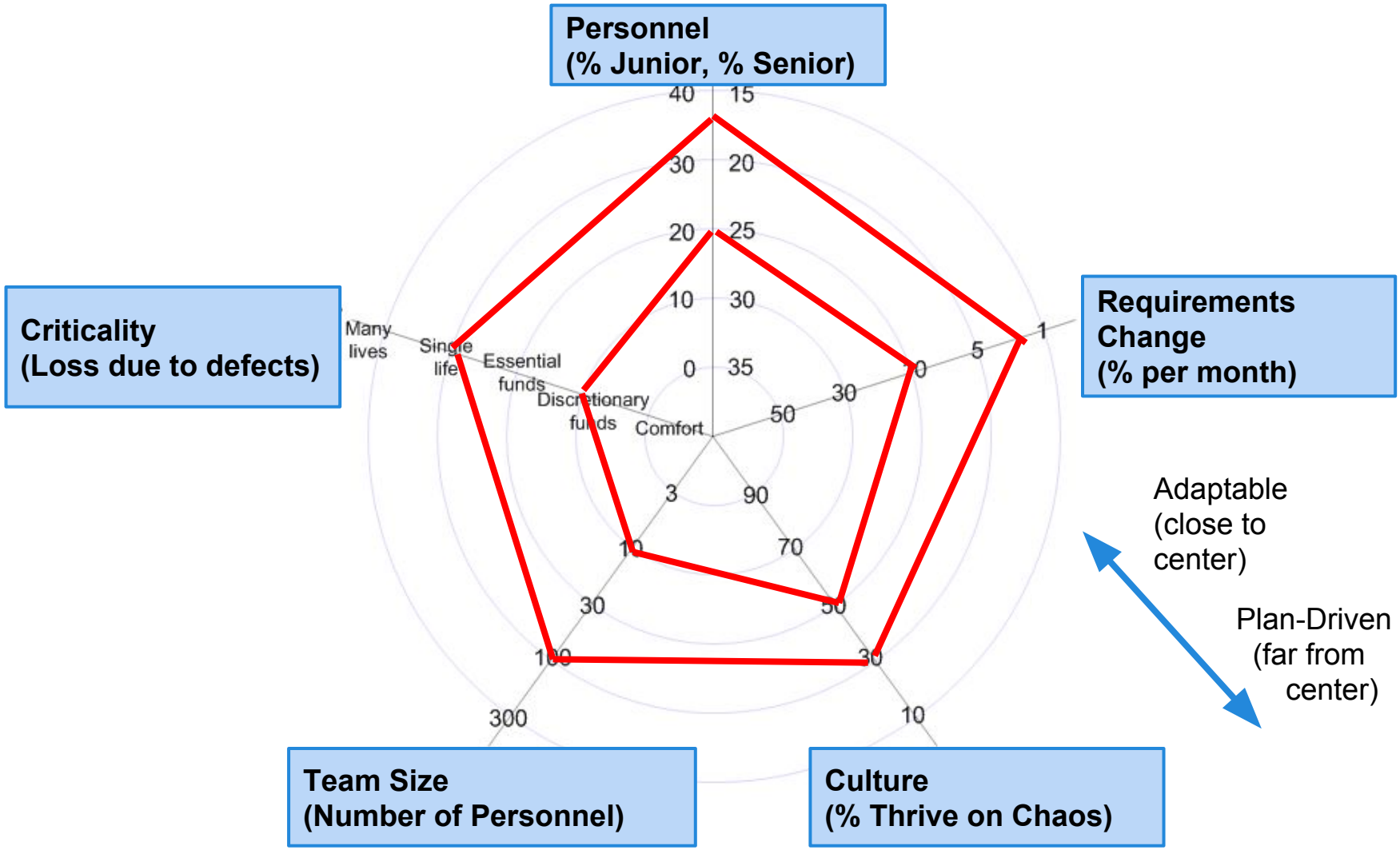
What are the advantages of the scrum process?

- (see iterative model, but also...)
- Very fast response to requirements change.
- Team members can choose how they approach their development responsibilities, as long as they still meet team goals.

What are some of the disadvantages?

- Requirements are unstable, and are not given enough importance.
- Often results in a lack of a design document or documentation.
- Relies on a good scrum master to keep meetings productive, and relies on the communication skills of the team members.

When to Choose Agile



Exercise

Given the following details about the project:

- Product installed on customer machines (not web-based)
- 20-year minimum product life
- Pressure to release early and update frequently
- Experienced developers
- Globally-distributed organization of about 300 developers
- Outages cost customers > $300K per hour
- High levels of technology and requirements uncertainty

What process would you select and why?

(you can combine elements of processes if you want)

Recap - Reasons for Process

We want a process because we are afraid that:

- The project will produce the wrong product.
- The project will produce a bad product.
- The project will be late.
- We all have to work 80 hour weeks.
- We will have to cut features.
- We will not have any fun working.

eXtreme Programming

Set of practices and principles to guide teams in the face of changing requirements.

- Enables **Adaptability**
  - in the business, technology, and team.
- Ensures **Predictability**
  - in plans and schedules, incorporating feedback and project tuning.
- Offers **Options**
  - Change direction or priorities at any time.
- Maintains **Humanity**
  - Focus on the idea of sufficiency.

XP Rules

There are rules you must following during development regarding:

- Planning
- Managing
- Designing
- Coding
- Testing

XP Rules - Planning

- User stories must be written.
  - Informal usage scenarios used to as requirements, to estimate implementation time, and to create acceptance test cases.
- Make frequent, small releases.
  - Tested, working software every two weeks.
- Divide the project into iterations.
  - Allows progress tracking.
  - Break stories into programming tasks. Do not put too many tasks into one iteration, instead reformulate iteration plan.

XP Rules - Managing

- Give an open, dedicated workspace.
  - Removes any barriers to communication.
  - Encourages people to work together, and increases community ownership of all project code.
- Set a sustainable pace.
  - If an iteration will not be finished on-schedule, remove tasks and reduce the scope.
- Vary developer tasks.
  - Avoid knowledge bottlenecks with well-rounded developers.
- Fix your process when it breaks.

XP Rules - Designing

- Simplicity is key.
  - Testable, understandable, browsable, and explainable.
- Create simple programs to prototype potential solutions.
- Never add functionality early.
  - This clutters up the system, and might end up being useless once requirements change.
- Refactor mercilessly.
  - Remove redundancy, eliminate unused functionality and improve obsolete designs.

XP Rules - Coding

- The customer should always be available.
  - Considered a member of the team.
  - Provides feedback, detailed requirements for programming tasks, help with test data.
- Always write tests before coding.
  - Solidifies requirements, gives developer a chance to think through their design.
- Program in pairs.
  - More eyes on the code will result in better code.
- Continuously integrate code into the project.
  - Everybody should be working with the latest code.

XP Rules - Testing

- All code must have unit tests.
  - Untested code is not acceptable for release.
  - Tests stored in repository along with code.
  - Tests enable refactoring and integration.
- All code must pass unit testing before it can be released.
- When a bug is found, create tests to prevent it from coming back.
- Create acceptance tests from user stories.
  - Run them often and publish the score.

Why is it Extreme?

Because we take good practices to extreme levels (turn the knob to 11):

- If code reviews are good, review code all the time (pair programming).
- If testing is good, test all the time (unit testing) - even the customers (acceptance testing).
- If design is good, make it part of daily business (refactoring).
- If simplicity is good, always leave the system with the simplest design that supports functionality

We Have Learned

- Processes give us control over development, focus developers, and the ability to mitigate risks.
- The waterfall model is plan-driven, good for projects with costly consequences. Focus on one, near perfect release.
- Agile methods (scrum, iterative) allow rapid changes to respond to customers' needs. Focus on rapid, "good enough" releases.
- eXtreme Programming advocates a set of development practices that may result in better software.

Next Time

- Requirements
  - The fine art of deciding what the $%#$ to build.
- Reading: Sommerville, chapter 4.

- Homework:
  - Get team selection in.