CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

DIT092 - Introduction to Agile Development

Gregory Gay

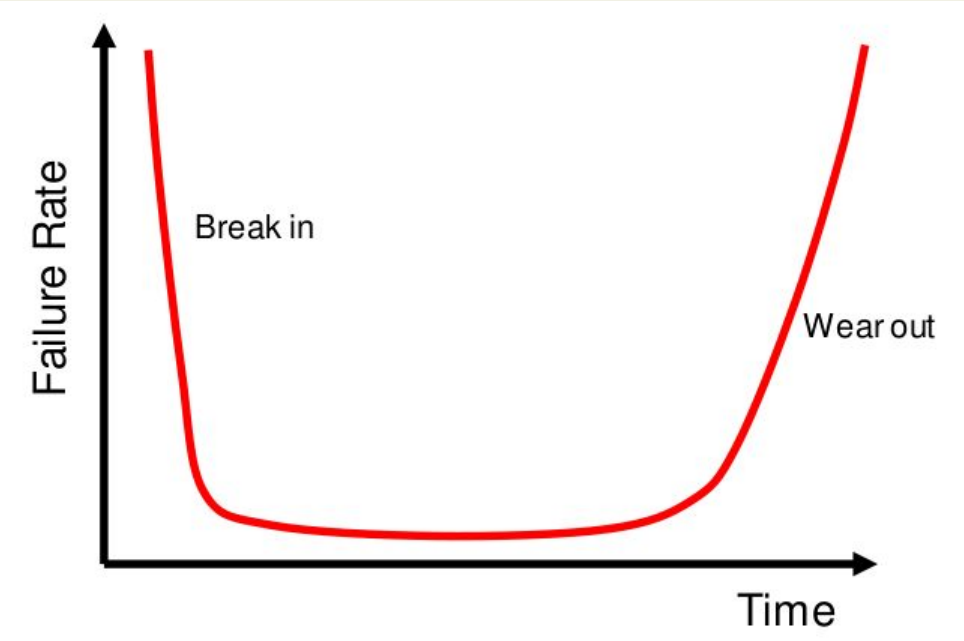
Today's Goals

- Understand risk factors in software development.
- Introduce software development processes
  - Processes and process models
  - Choosing a process
    - AKA: planning and risk management
  - Waterfall Model
  - Agile Model
- Cost estimation (planning poker)

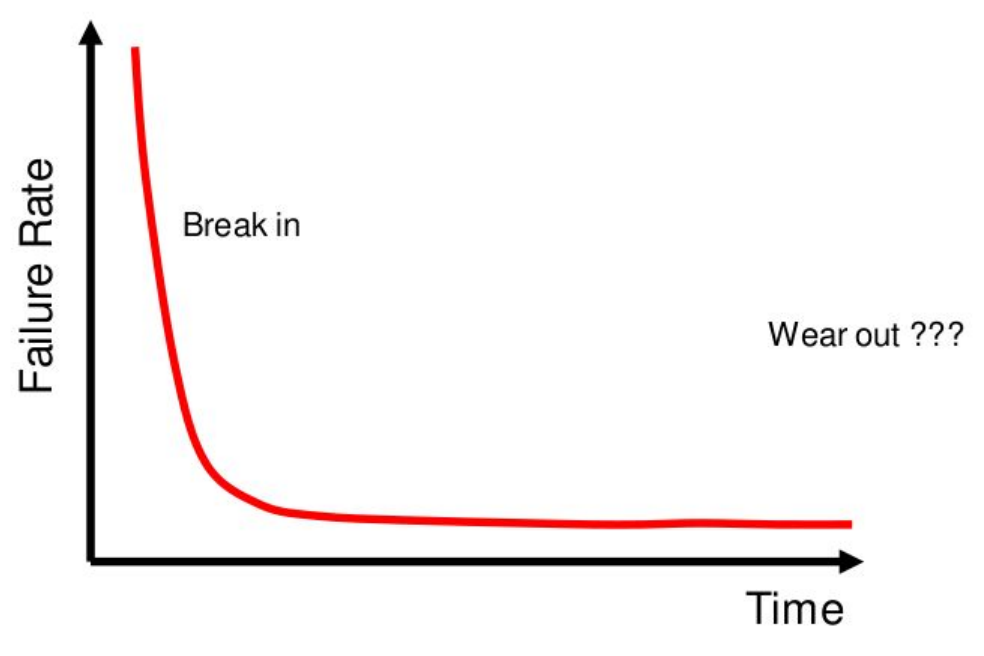
Development is %S##% Hard

- Complexity
  - More complex than a skyscraper.
- Changeability
  - Software is “easy” to change.
- Invisibility
  - We cannot see the progress of development.
- Conformity
  - Software must be molded to fit external constraints.

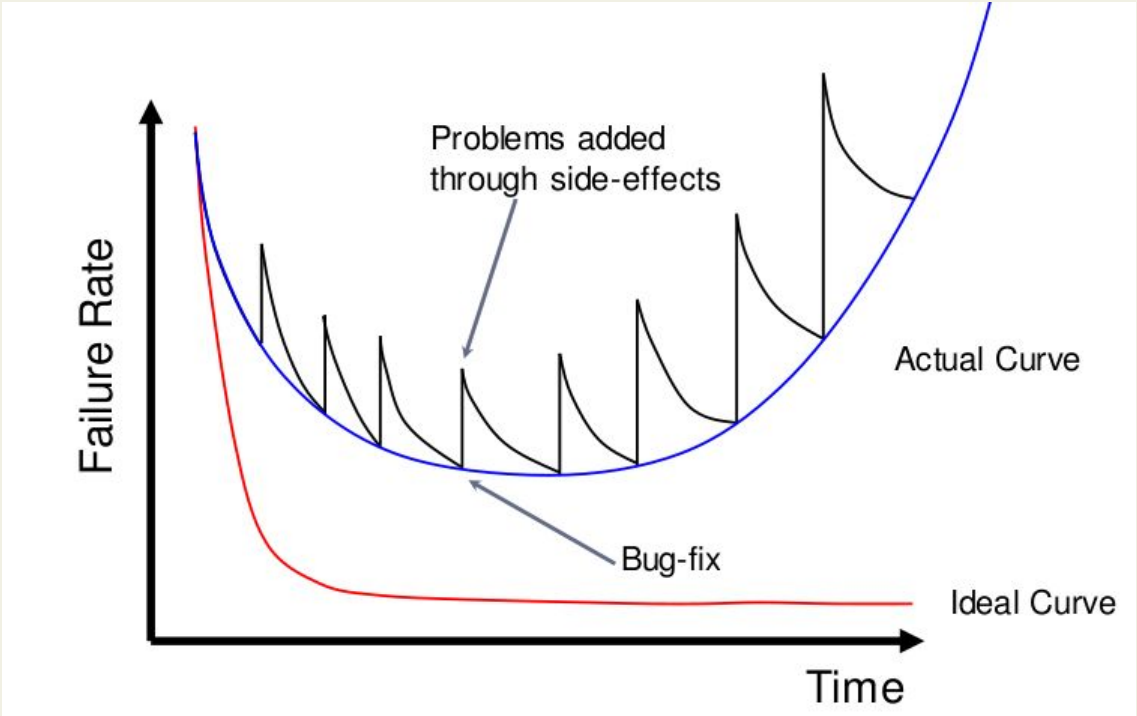
Hardware Failure



Software Failure (We Think)



Software Failure (In Reality)



Prevailing Paradigm:
Code and Fix

Life Cycle of Software

Any product has a life cycle: a timeline that can be split into the required phases of existence.

What are the phases of a software lifecycle?

Project Planning

Requirements Definition

Software Design

Implementation

Testing

Release

Operation and Maintenance

Why Are We Stuck In Code & Fix?

We know there are activities that must be performed:

- Specification, Design, Coding, Testing, Evolution

… But **we lack structure and guidance**:

- When are we done? When do we move on?
- Activities must be planned and modeled if they are to be managed.

Asking Questions

Development begins by asking questions.

1. Why is the system being developed?
2. What will be done?
3. When will it be accomplished?
4. Who is responsible?
5. Where are they located within the organization?
6. How will the job be completed (technically and managerially)?
7. How much of each resource is needed?

Risk Management

The principle task of a manager is to minimize (avoid or mitigate) risk.

- The “risk” in an activity is a measure of the uncertainty of the outcome of that activity.
  - Risk is related to the amount and quality of available information.
  - What are the risk factors? What will be their impact? How likely are they to arise?

Typical Engineered Systems

What do these have in common?

Defining a Process

Process: a flow of events that describes how something works.

- In our case - defines a timeline of activities required to build software.
- Structures who is doing what, when, and how.

Risk Management

High-risk activities cause schedule and cost overruns.

A visible process provides the means to track, assess, and mitigate risk.

Processes provide quality and predictability by removing risk.

Engineering Process Model

Set of sequential, discrete phases:

- Specification
  - Set out requirements and constraints.
- Design
  - Produce a paper model of the system.
- Manufacture
  - Build the system.
- Test
  - Check the system meets specifications.
- Install
  - Deliver the system to the customer.
- Maintain
  - Repair faults over time.

Software Process Models

Why is software different from other products?

- Specifications are often incomplete and vague (complex functionality, intangibility)
- Blurred distinction between specification, design, and manufacture phases.
- No physical realization of the system for testing.
- Software does not wear out.

Process Characteristics

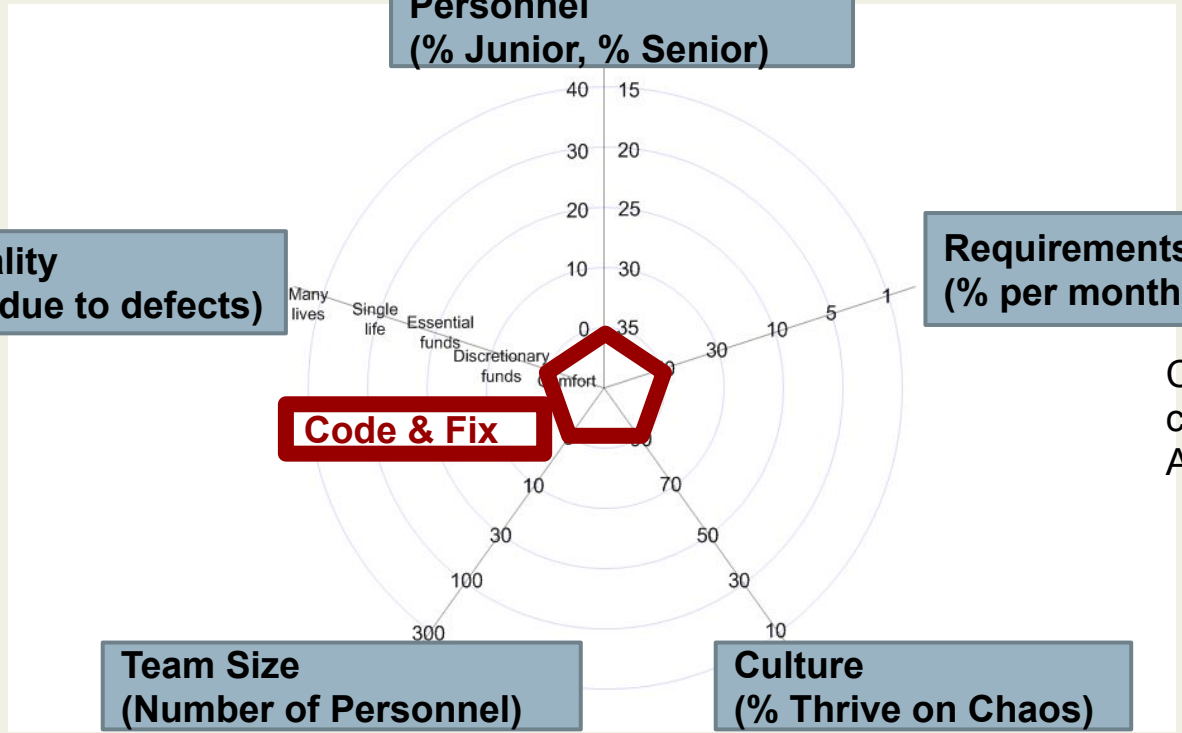
What are characteristics of a good process?

- Understandability
  - Is the process defined and understandable?
- Visibility
  - Is the progress of the process externally visible?
- Supportability
  - Can the process be supported by tools?
- Acceptability
  - Is the process acceptable to those involved in it?

Process Characteristics

- Reliability
  - Can we discover process errors before they result in product errors?
- Robustness
  - Can the process continue despite unexpected problems?
- Maintainability
  - Can the process evolve to meet organizational needs?
- Rapidity
  - How fast can the system be produced?

Considering Developmental Factors



Choosing a Process

How do we choose a process?

- Consider project characteristics.
- Consider project risks and how they can be mitigated.
- Determine the degree of rigor required.
- Define a task set for each development phase.
  - Task set = {engineering tasks, work products produced, quality assurance, schedule of project milestones}

There is no one-size-fits-all software process.

Code and Fix Model

- Short, interleaved specification and design phases.
- Interleaved implementation, testing, and maintenance phases.

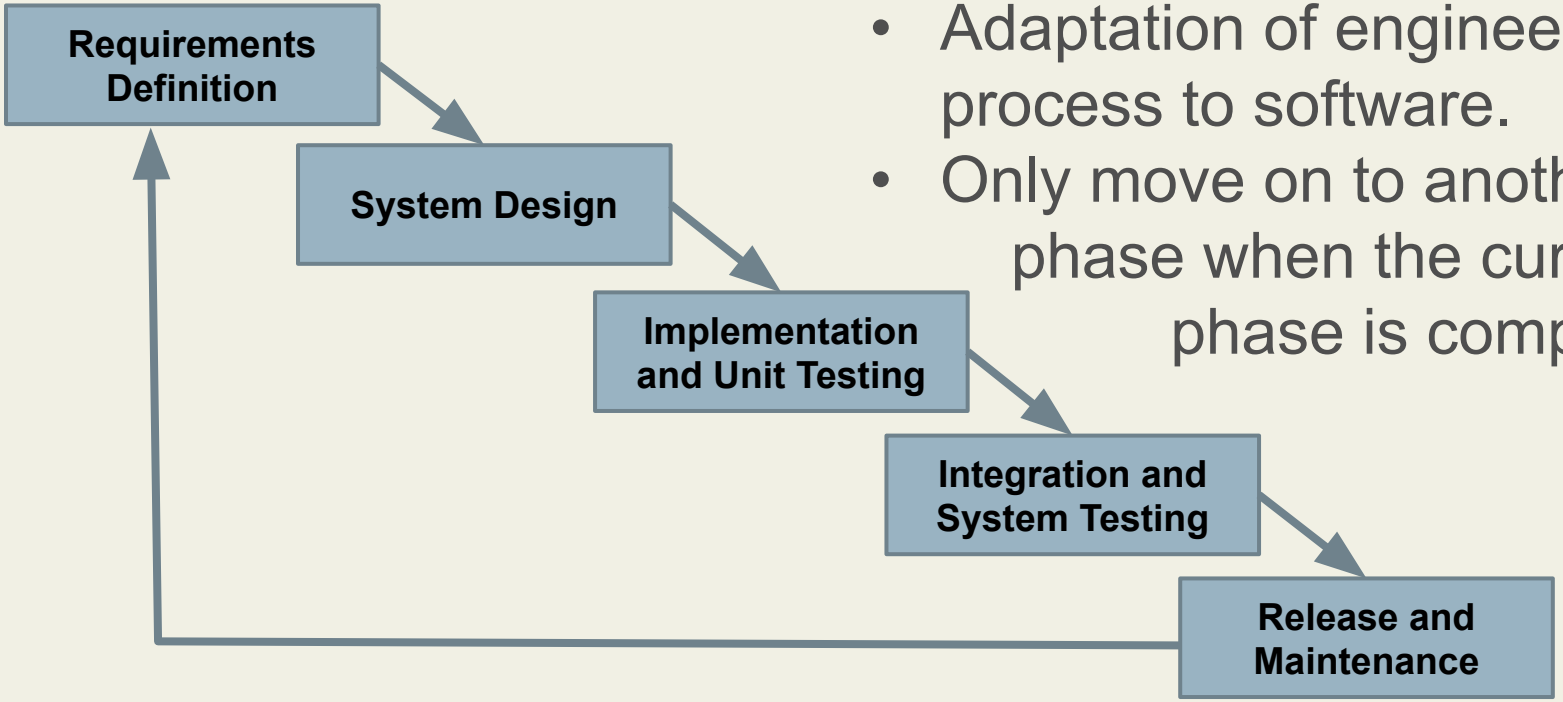
Applicability

- Useful for small, simple projects.
- Allows fast time-to-market.

Potential Problems

- Quality
- Maintainability

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.

Waterfall Model Task Set

Phase	Output Document
Product Conception	Feasibility Study
Requirements Definition	Requirements Document
System Specification	Functionality specification, acceptance test plan, draft of user manual
Architectural, Interface, and Detailed Design	Architectural, Interface, and Detailed Design Specifications. System, Integration, and Unit test plans
Coding	Source Code
Unit, Module, Integration, and System Testing	Testing Reports
Acceptance Testing	Final system and documentation

Applicability

What type of projects should use Waterfall?

- Projects where the requirements are stable.
- Projects where impact of failure is severe.
- Projects with high turnover or inexperienced developers.

Problems

- It is hard to know when you are done with a phase.
- Inflexible model that **does not accommodate change**.
  - Hard to respond to unexpected risk.
  - What if the users hate your final system?
- You ***need*** to return to earlier phases as details change.
  - You rarely know your requirements that early.
  - Implementation details often emerge during implementation.
- You do not generally seek feedback until the end.
  - And often **can't** - hard to show a partial demo.

Let’s take a break

From The “Creator” Of Waterfall...

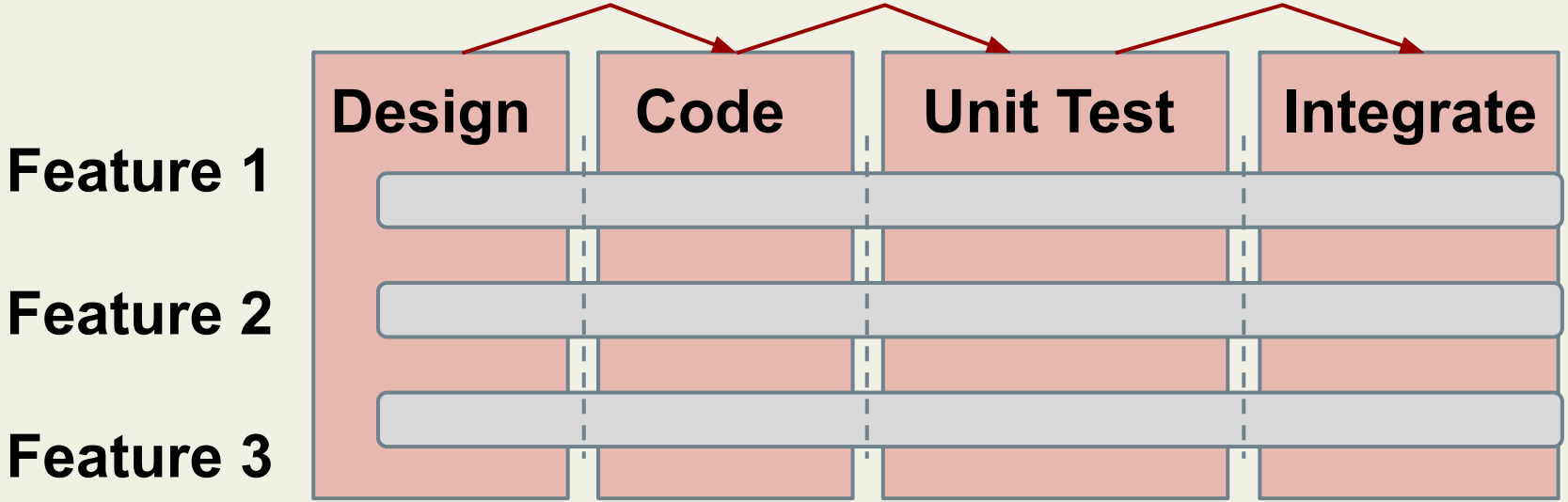
“I believe in the concept, but the implementation is risky and invites failure.”

- Winston Royce

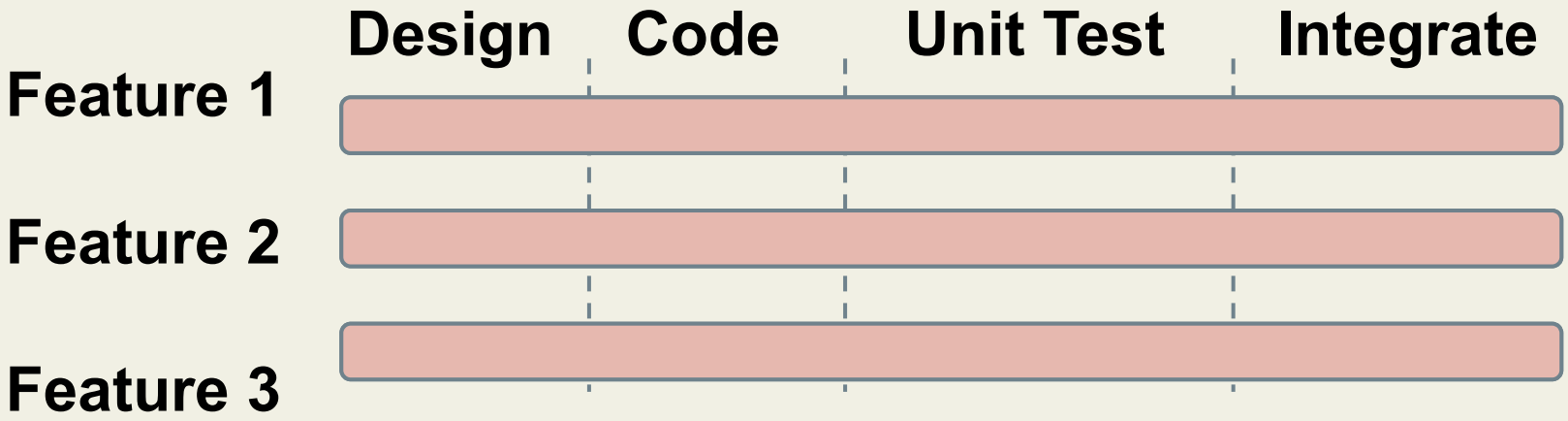
Enter… Agile Development

- “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.

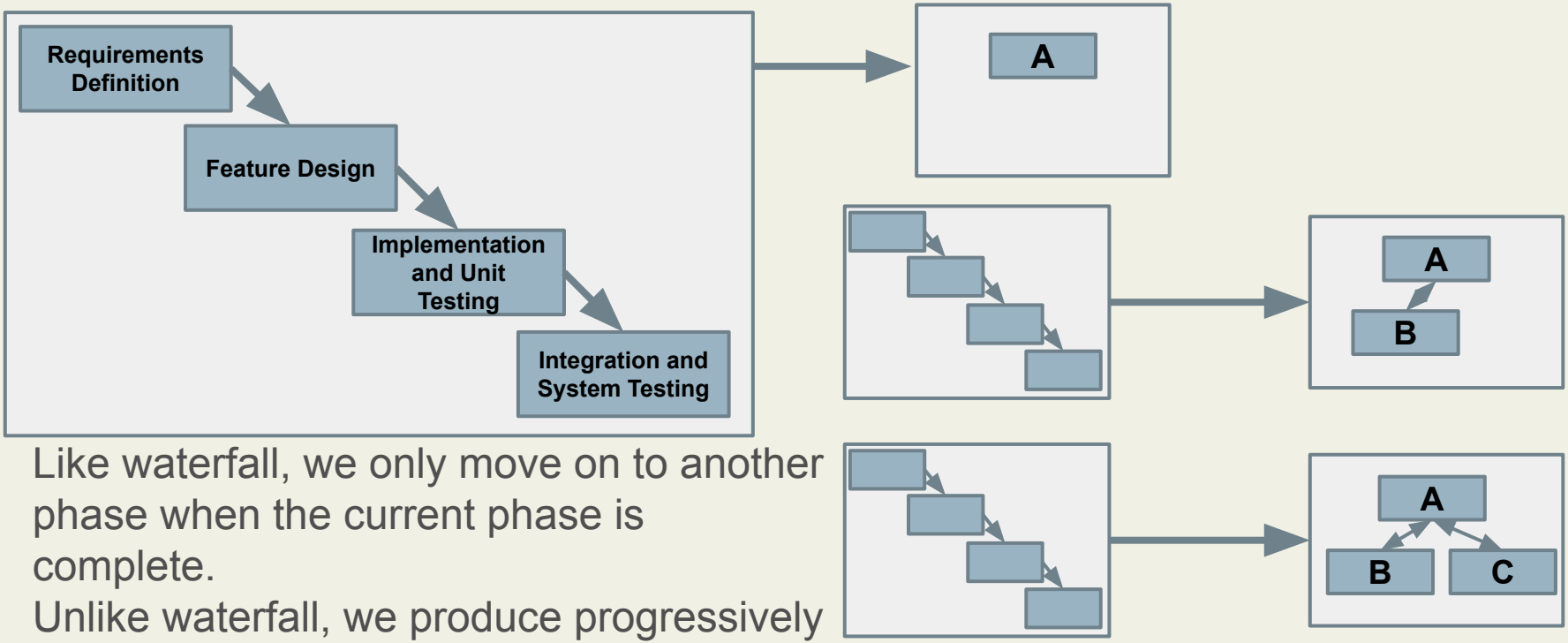
Work Partitioning: Waterfall



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?

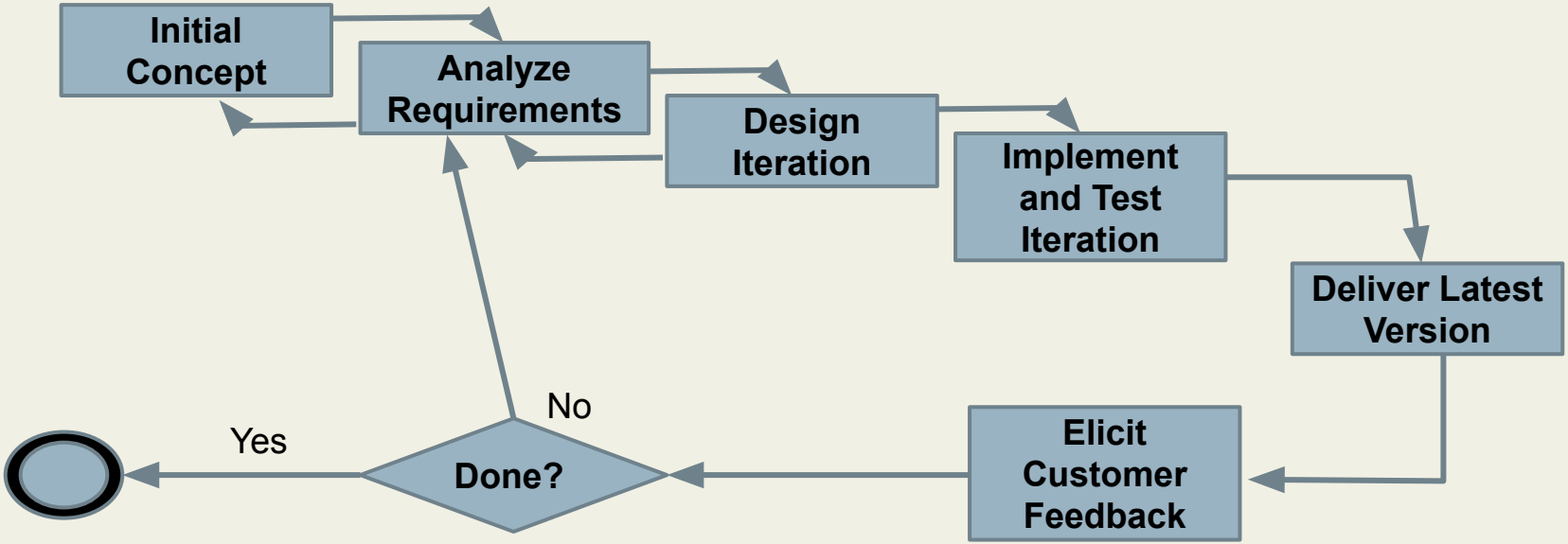
- Customers can start using the system earlier.
  - … and feedback can be obtained more frequently.
- Slower integration of features allows easier testing of individual features and feature interactions.
- If time/budget runs out, partial product can be released.

The Incremental Model

What are disadvantages of incremental development?

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

The Iterative/Evolutionary Model



Aren't incremental and iterative the same?

- **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 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 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.

The Iterative Model

What are the disadvantages of iterative development?

- "Good enough" is ***very*** risky if software problems can harm humans or their operating environment.
  - You still need planning, detailed design if there is risk
- Frequent releases often results in rushed releases.
  - Building on bad foundations results in a bad product.
  - Do not rush cycles

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.
  - Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently.
- Business people and developers must work together daily throughout the project.

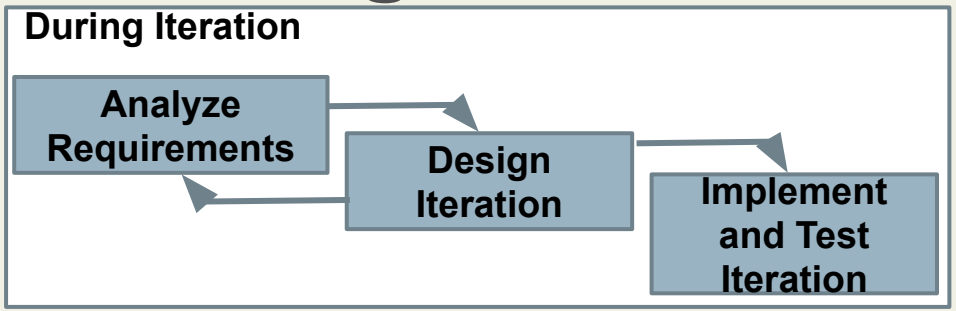
Agile Principles

- 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 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.

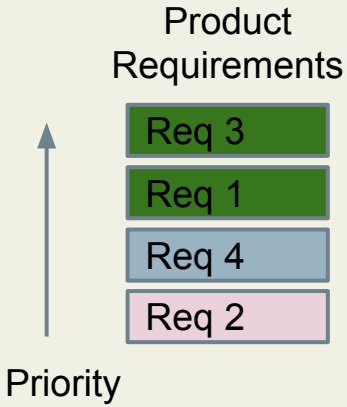
Agile Principles

- Continuous attention to technical excellence and good design enhances agility.
- Simplicity is essential.
  - The art of maximizing the amount of work not done
- 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.

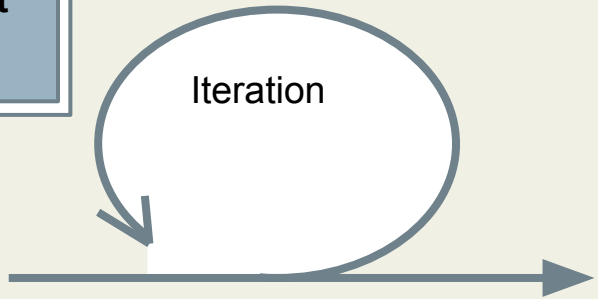
The Agile Model

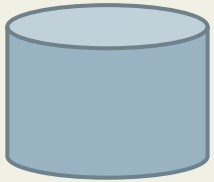


Agile is not ad-hoc. An iteration should have some kind of structure.



Requirements Scheduled for Iteration



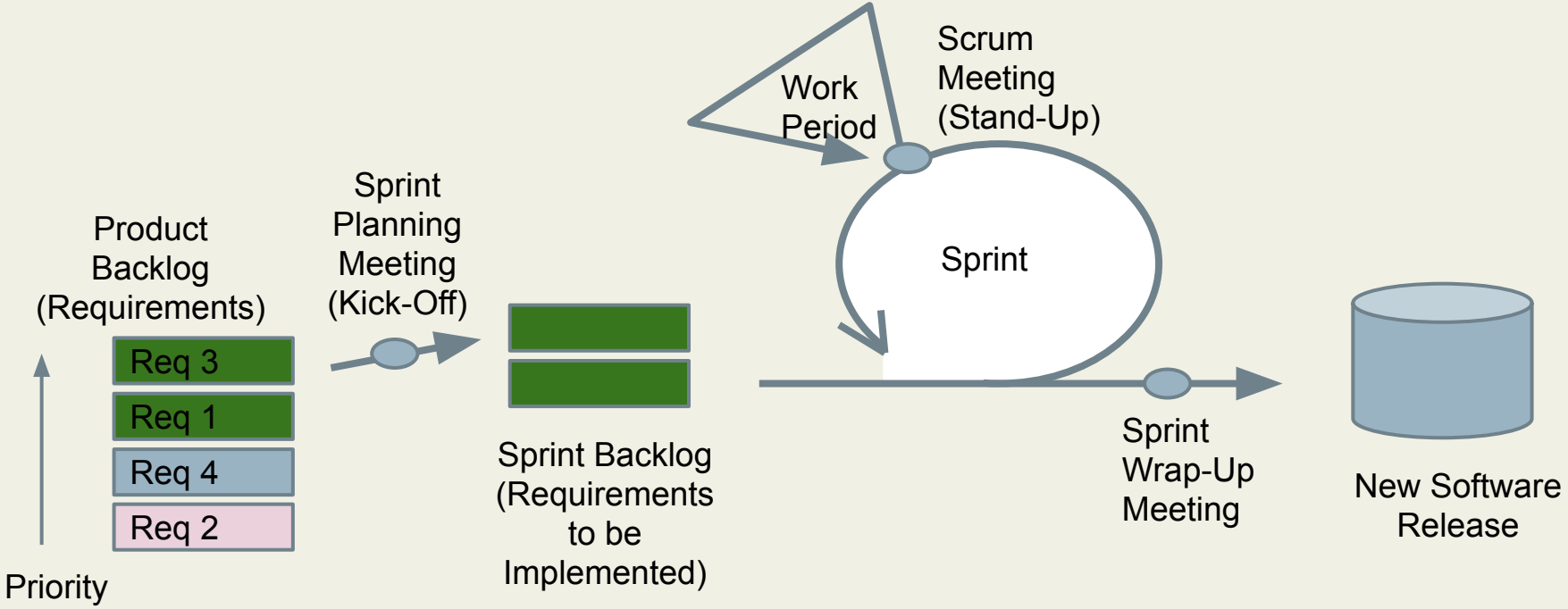


New Software Release

The Scrum Model

- Bring the team together to discuss problems as a team.
  - Then send them out to accomplish their individual goals.
- Individuals define their own work 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 to customers and communicates feedback to the team.
  - Maintains and prioritizes the requirement backlog
- **The Scrum Master**
  - The team “coach”.
    - Removes impediments to success.
    - Facilitates meetings and communication.
    - Mediates 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?

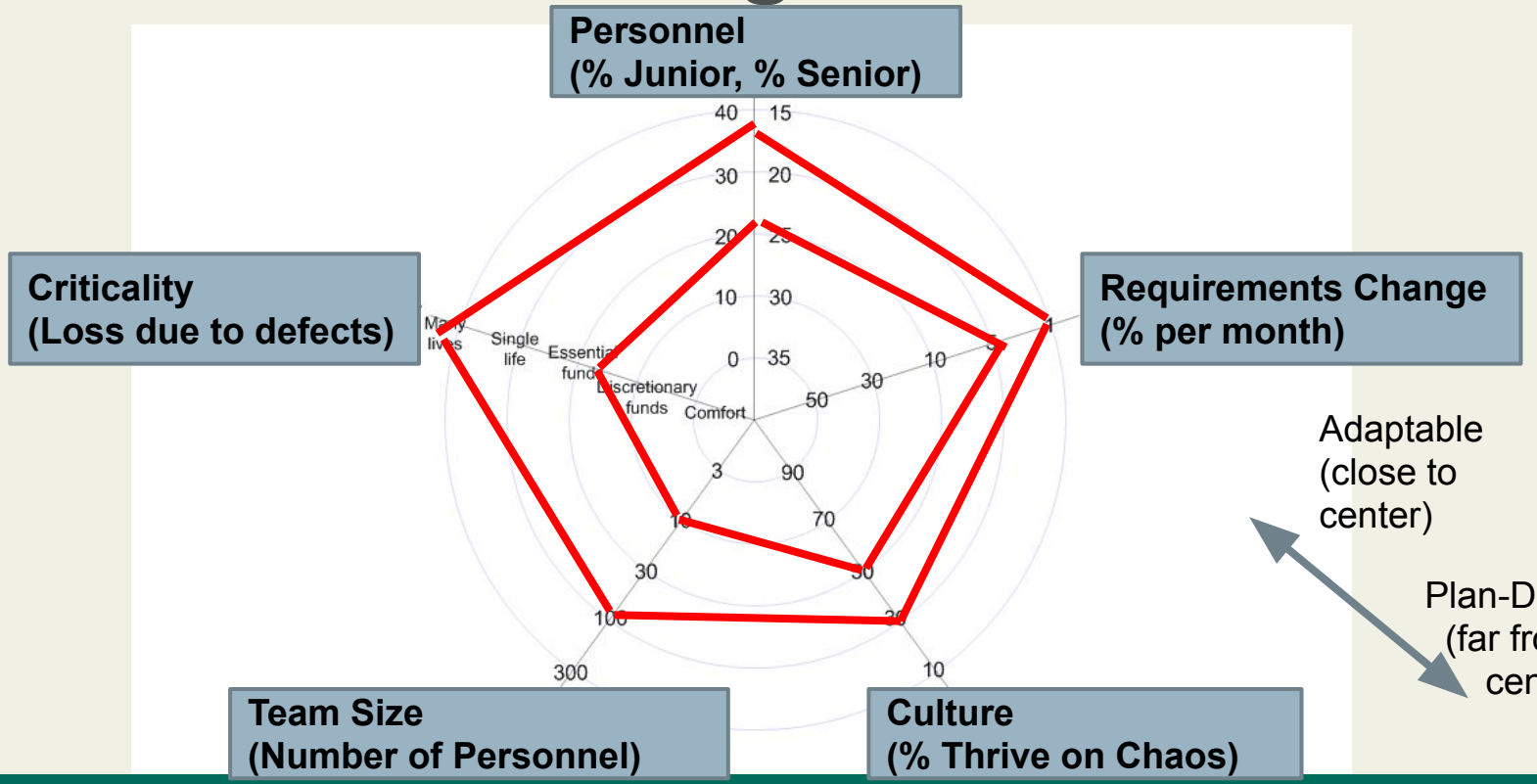
- (see iterative model, but also...)
- Very fast response to requirements change.
- Team members can choose how they approach their development responsibilities, as long as they still meet team goals.

The Scrum Model

What are the disadvantages of the scrum process?

- Requirements are unstable, and may not be given enough importance.
- Often results in a lack of a cohesive 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



Discussion

Given the following details about the project:

- Installed on customer machines (not web-based)
- 20-year intended product life
- Pressure to release early and update frequently
- Experienced developers
- Globally-distributed organization of 300 developers
- Outages cost customers > 3M kr per hour
- High levels of technology and requirements uncertainty

What process would you select?

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 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 as requirements, to estimate implementation time, and to create 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.

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.
  - Well-rounded developers avoid knowledge bottlenecks.
- Fix your process when it breaks.

XP Rules - Designing

- Simplicity is key.
  - Testable, understandable, browsable, and explainable.
- Create simple programs to prototype 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.
  - Provides feedback, detailed requirements for programming tasks, help with test data.
- Always write tests before coding.
  - Solidifies requirements, think through 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 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).
- 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

Cost Estimation Using Planning Poker

Project Planning

- We need to know:
  - How long is the project going to take?
  - How much is it going to cost? (how much effort?)
- There is no magic formula!
  - … But we can make educated guesses in order to manage risk.
  - Estimation gives us the ability to schedule and allocate resources and workload.

The Product Backlog

- To-do list of project features.
  - Each sprint completes 1+ features.
  - Can be prioritized.
  - Can estimate how long it will take to complete them.
- Each feature has:
  - **User Story:** Explains how user interacts with the feature.
  - **Tasks:** The steps to be completed to deliver the feature.
  - **Acceptance Criteria:** How do we know whether the feature works?
  - **Story Points:** Estimate of effort needed to complete.

Story Points

- Estimate of the amount of work needed to complete a feature.
- Based on Fibonacci sequence: 1, 2, 3, 5, 8, 13, 21.
  - Abstract unit. Not convertible to time or cost.
  - Based on intuition (X% of 5 point stories are completed in four weeks).
- Extrapolate **velocity** as story points per sprint.

Planning Poker

1. Developers are each dealt an identical hand of Estimation Poker Cards.
   Cards: 1, 2, 3, 5, 8, 13, 21, coffee (break) and question mark (unknown).
2. The product owner presents the feature.
3. The development team discusses the feature.
4. Developers secretly select a card reflecting their individual assessment of the feature.
5. On count of three, the developers share their card.
6. If disagreement, discuss why the story is seen differently by the developers who played the highest and lowest cards.
7. Repeat 4-6 until there is consensus.

Why Planning Poker?

- Simultaneous reveal encourages team members to think independently.
- Encourages discussion of disagreements.
- Focus on perspectives that differ the most.
  - Conversations focus on highest and lowest estimate.
- “Story Points” reflect a relative scale, not complex formulae.
  - We may lack deep design detail.
  - Relative complexity only needs broad outline.

Calculating Velocity

- Shows amount of work a team can do per sprint and predicted date for finishing the project.
- Requires completing initial sprints.
  - Finished 75% of features (15/20 points) in sprint 1.
  - Finished 100% of features (18/18 points) in spring 2.
  - Velocity = (15+18)/2 = 16.5
- Velocity = Average Story Points Per Sprint
  - Lets you predict how many features to plan per sprint.

Estimating Schedule

- Story points and velocity can be used to predict when you will finish.
  - Want to complete four features (5, 13, 3, 8) = 29 points.
  - Velocity of 10 points per sprint.
    - 95% of the time they hit 10.
  - Will need 3-4 sprints.
    - Start by May 1 for June 1 release.
    - If not ready by June 1, need to wait for September 1.
      - Is it worth the delay? Can we just release without a feature?
      - 13-point story is a problem. Can this be split into two smaller features?

Estimating Budget

- Budget = (Story Points / Velocity) * (Team Cost Per Sprint) + (Other Costs)
  - (Team Cost Per Sprint) = Hours per Sprint * Team Members * Cost Per Hour
- Example:
  - Budget = 1500000 kr = 5 * 200000 + 500000
    - 100 Story Points, Velocity of 20 = (100/20) = 5 sprints
    - 80 hours per sprint, 5 team members, 500 kr per hour = (80 * 5 * 500) = 200000 kr per sprint
    - 500000 kr non-labor cost.
    - (Confidence interval: 1200000 - 1800000)

Try Planning Poker

Summary

- 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 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.
- Planning Poker and Story Points can be used to estimate cost and schedule.

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY