Software Processes

Week 1

Sometimes we make the process more complicated than we need to.

— Joseph B. Wirthlin

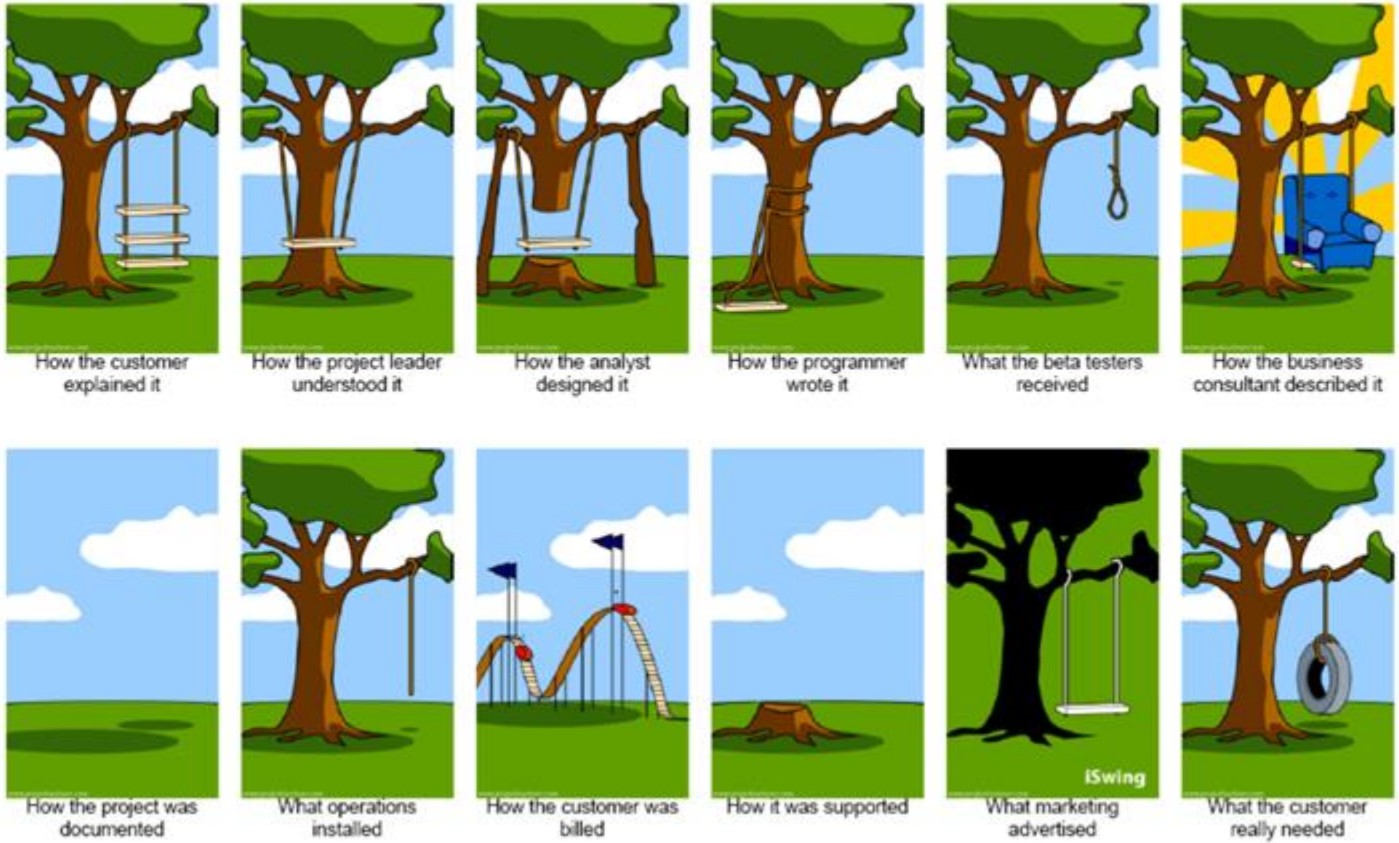
Objective

- Understand the life cycle of software development
- Understand various software processes and their advantages and disadvantages
- Understand that there is no one-size-fit-all software process

Contents

- Systems Development Life Cycle (SDLC)
- Various software process models
  - Waterfall model and its early extensions
  - Agile manifesto
  - Extreme programming
  - Scrum

How Software Project Really Works!



How the customer explained it

How the project leader understood it

How the analyst designed it

How the programmer wrote it

What the beta testers received

How the business consultant described it

How the project was documented

What operations installed

How the customer was billed

How it was supported

What marketing advertised

What the customer really needed

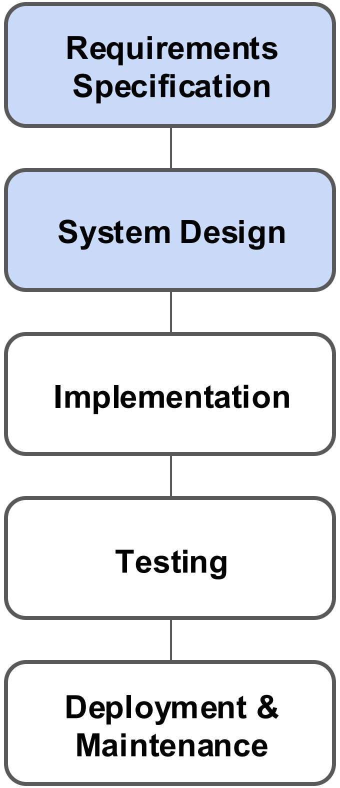
Systems Development Life Cycle (SDLC)

- Typical simplified 5-phase description
  - Requirements specification
  - System design
  - Implementation
  - Testing
  - Deployment & Maintenance

- Phase division can vary by:
  - Process models
  - Usages

Phases in SDLC

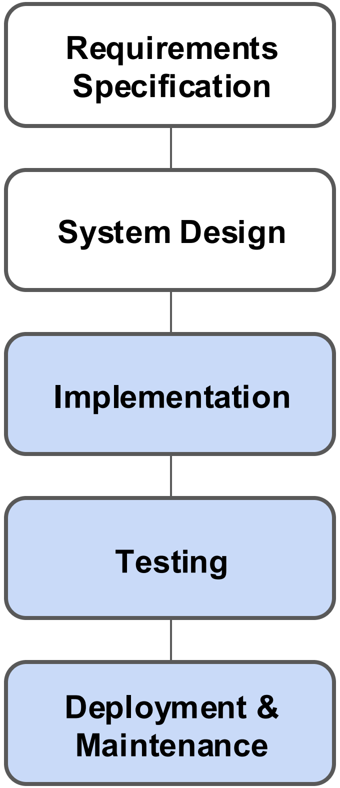
- **Requirements Specification**
  - Preceded by market research
    - Customers do not know what they want
  - Define the product
- **System Design**
  - Specification of architecture, data structure, algorithmic details, etc.
  - Describe the plan rigorously



Phases in SDLC

- **Implementation & Testing**
  - Write the code (~15% of total cycle)
  - Test and integrate (~50% of total cycle)
  - Document the internal designs

- **Deployment & Maintenance**
  - Package product into a product or service
  - Detect and fix bugs



Software (Development) Process Models

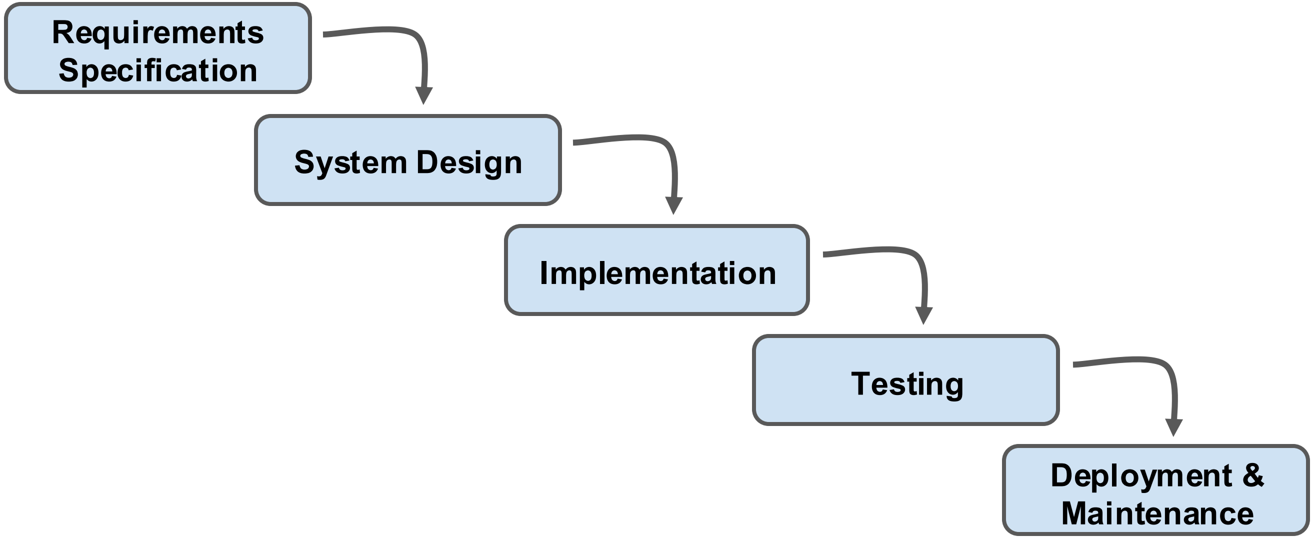
- **Abstract representation** of the development process
- Describes *what, when and how* to do in each phase software development life cycle
- Having a suitable software development process is essential for the success of projects

Software (Development) Process Models

- 1970s
  - Waterfall, Prototyping, Iterative & Incremental Development
- 1980s ~ early 1990s
  - Spiral Model, V-Model
- Mid 1990s ~ now
  - Rational Unified Process (RUP), Extreme Programming (XP), Scrum
  - Typically referred to as *agile methodologies*, after the **Agile Manifesto** published in 2001

Waterfall Model[1]

- A *rigid sequential* design process
  - Each stage completes before the next starts
  - Assumes plan is clear from start



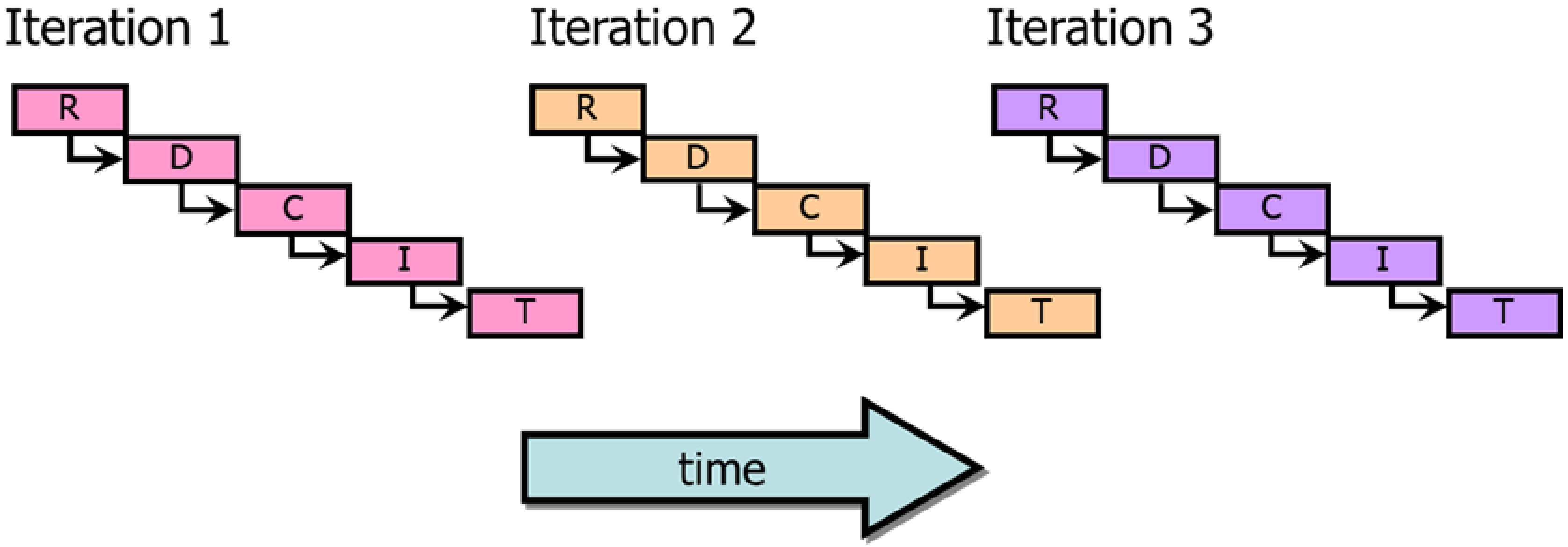
[1] Royce, W.W. Managing the Development of Large Software Systems. Proceedings of IEEE WESCON, 26 (1970), 328-388

Waterfall Model

- Advantages
  - Easy-to-follow: good for inexperienced and/or new workers
  - Easy to estimate the resource (cost & time) usage
  - Strong control over the process (emphasis on documentation)
  - Scalable to large projects
- Drawbacks
  - Requires perfect planning
  - Hard to reflect rapid market changes

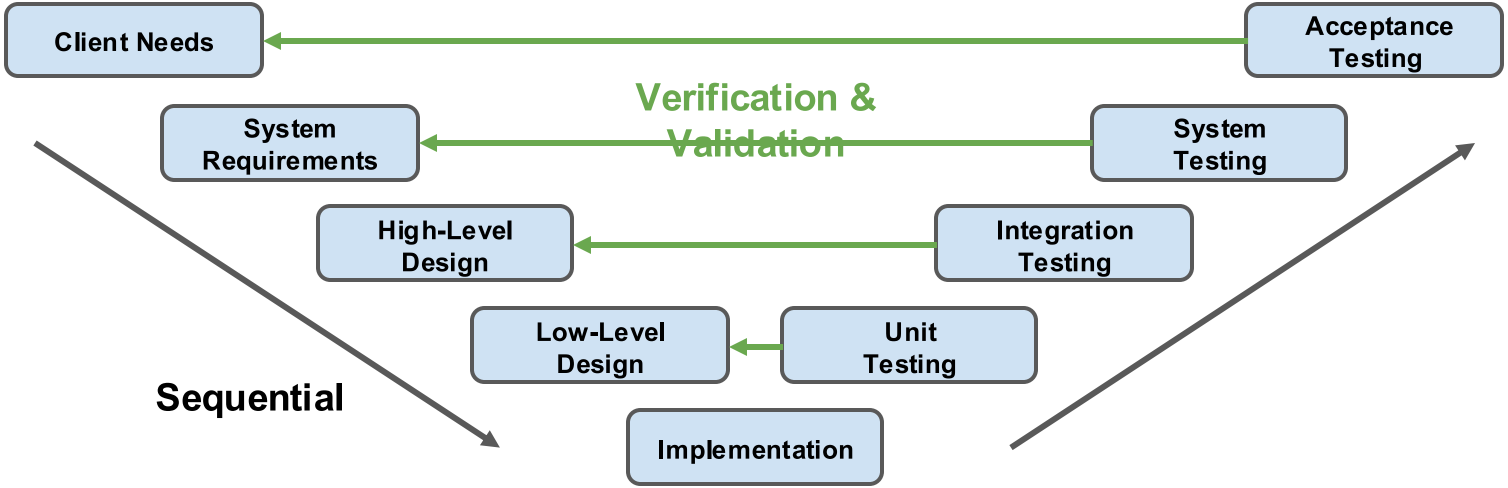
Early Extensions: Iterative Model

- Each iteration includes the whole cycle
- Feedback at the end of each iteration



Early Extensions: V-Model[1]

- Multiple testing associated to each level of abstraction
  - Enables partial reuse
  - Higher guarantee of success

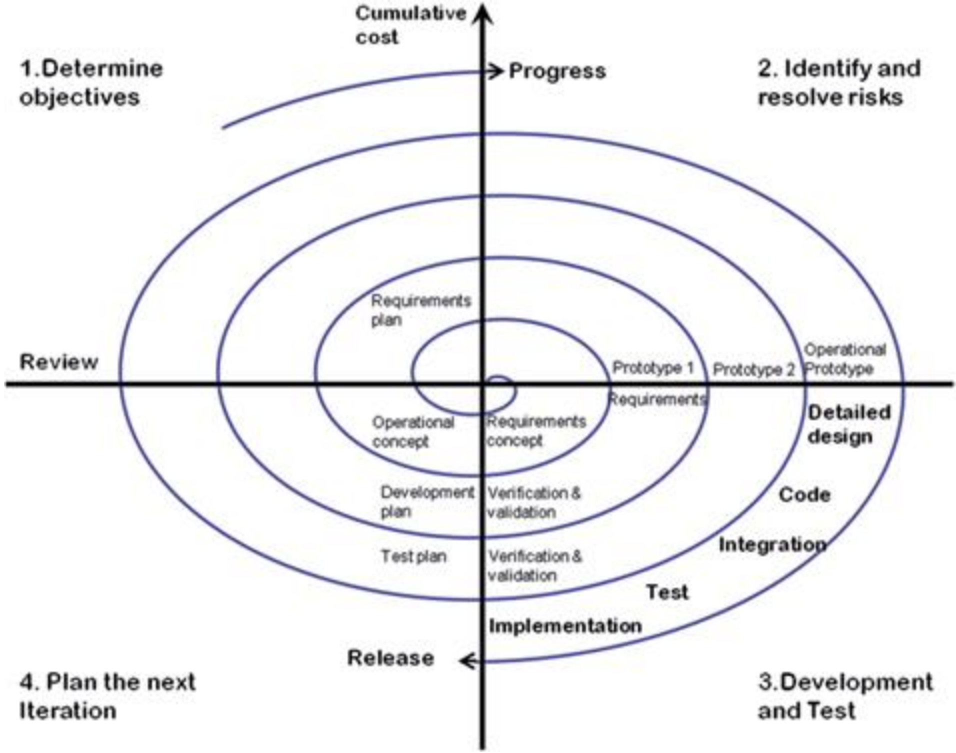


[1] Forsberg, Kevin, and Harold Mooz. "The relationship of system engineering to the project cycle." *INCOSE international symposium*. Vol. 1. No. 1. 1991.

Early Extensions: Spiral Model[1]

- Prototyping + Waterfall
  - Each iteration with 4 stages
  - Smaller scaled iterations
  - Separate risk analysis stage
  - Have working prototypes

- Disadvantages
  - Higher cost for risk analysis
  - Non-trivial to scope each iteration



[1] Boehm, Barry W. "A spiral model of software development and enhancement." *Computer* 21.5 (1988): 61-72.

Are the Extensions Enough?

- Extended models overcome the limitations of the waterfall model
- However, they do not meet rapidly changing user and market expectations while still requiring high level of expertise of the project manager

Agile Development Process

- A group of software development methods that promotes
  - Adaptive planning
  - Evolutionary development and delivery
  - Time-boxed iterative approach
  - Rapid and flexible response to change

- Example agile processes:
  - Agile Manifesto
  - Rational Unified Process (RUP)
  - Extreme Programming (XP)
  - Scrum

Agile Manifesto

- Published by 17 developers in February 2001[1], after discussion of lightweight development methods

"We are uncovering better ways of developing software by doing it and helping others do it. We 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****."*

[1] Beck, Kent, et al. "The agile manifesto." (2001).

Agile Manifesto: 12 Principles (1/3)

1. Satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development.
3. Deliver working software frequently, with a preference to the shorter timescale.
4. Work together daily throughout the project.

Source: Beck, Kent, et al. "Principles behind the Agile Manifesto" (2001).

Agile Manifesto: 12 Principles (2/3)

5. Build projects around motivated individuals. Give them the support they need.
6. The most efficient and effective method of of conveying information is face to face conversation.
7. Working software is the primary measure of progress.
8. Maintain a constant pace indefinitely.

Source: Beck, Kent, et al. "Principles behind the Agile Manifesto" (2001).

Agile Manifesto: 12 Principles (3/3)

9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. Those teams reflects on how to become more effective, and tunes the behavior at regular intervals.

Source: Beck, Kent, et al. "Principles behind the Agile Manifesto" (2001).

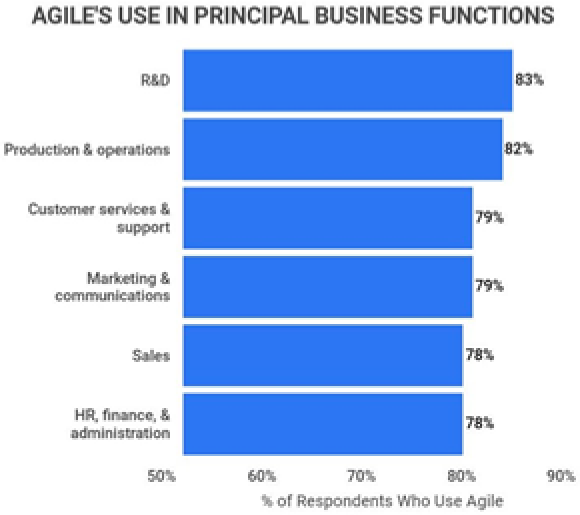
Waterfall vs. Agile

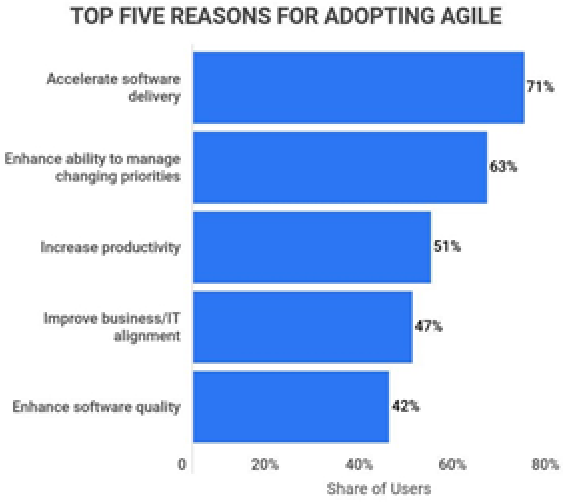
	Waterfall	Agile
Timeline	Fixed	Fixed (short term only)
Client Involvement	No	Promoted
Budget	Fixed	Flexible
Success Rates (Easy/Difficult Tasks)	High / Low (architect-centric)	High / High (member-centric)
Documentations	Well organized	Poor

*Others points: risk management, scalability, etc.

Waterfall vs. Agile

- Some statistics[1] (Nov. 2022)
  - At least 71% of U.S. companies are now using Agile
  - Success rates: Agile (64%) vs Waterfall (49%)

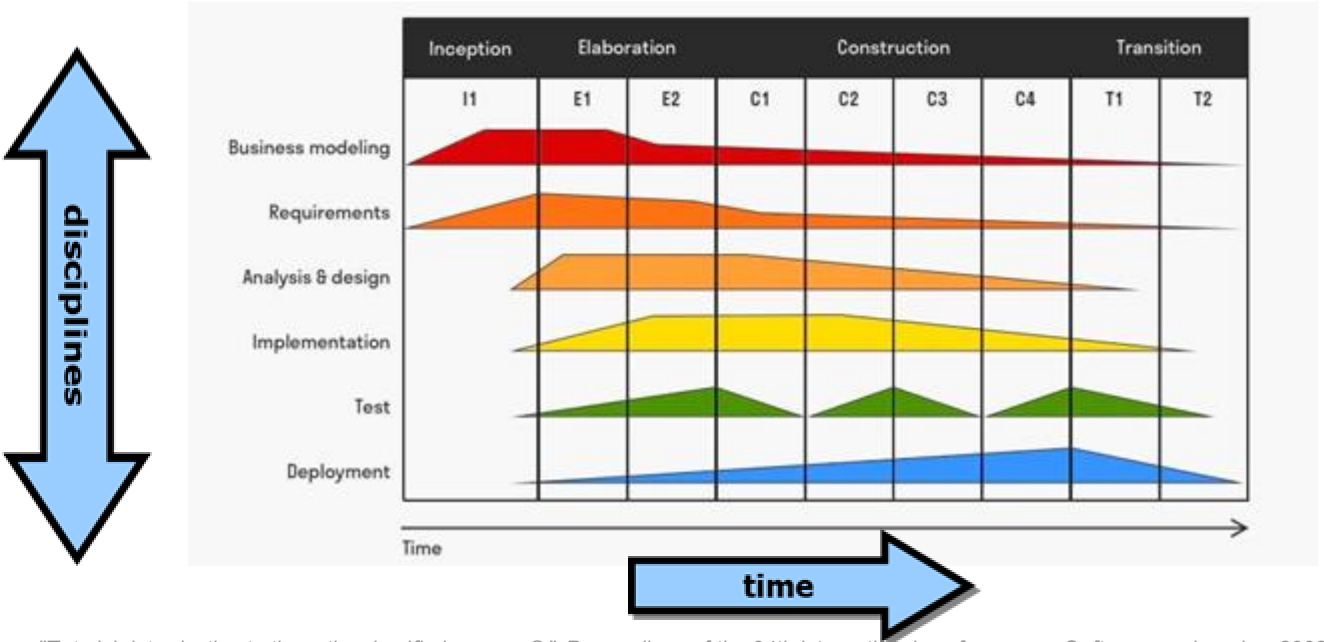

AGILE'S USE IN PRINCIPAL BUSINESS FUNCTIONS


TOP FIVE REASONS FOR ADOPTING AGILE

[1] https://www.zippia.com/advice/agile-statistics/#:~:text=Only%2027.4%25%20of%20manufacturing%20companies,combination%20of%20methodologies%2C%20including%20Agile.

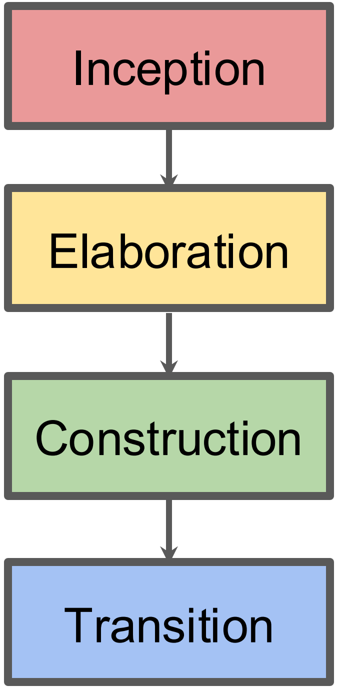
Rational Unified Process (RUP)[1]

- Development process by Rational Software Corporation*
- Introduced in 1996, later acquired by IBM in 2003



[1] Kruchten, Philippe. "Tutorial: introduction to the rational unified process®." *Proceedings of the 24th international conference on Software engineering*. 2002.

RUP Phases



Inception	Understand what product to build
Elaboration	Understand how to build the project
Construction	Build the project
Transition	Validate & deploy the product

Inception: Know What to Build

- Prepare vision document and initial business case
  - Include risk assessment and resource estimate
- Develop high-level project requirements
  - Initial use-case and domain models (10-20% done)
- Manage project scope
  - Reduce risk by identifying all key requirements
  - Acknowledge that requirements will change
- Manage change, use iterative process

Elaboration: Know How to Build It

- Detail requirements as necessary (~80% complete)
  - Less essential ones may not be fleshed out
- Produce an executable and stable architecture
  - Roughly 10% of code is implemented.
- Drive architecture with key use cases
- Verify architectural qualities
  - Reliability & Scalability
- Continuously assess business case, risk profile and development plan

Construction: Build the Product

- Complete requirements and design model
- Design, implement and test each component
- Build daily or weekly with automated build process
- Test each build
  - Automate regression testing
- Deliver fully functional software (beta release)
- Produce release descriptions

Transition: Deploy to Users

- Produce incremental ‘bug-fix’ releases
- Update user manuals and deployment documentation
- Update release descriptions
- Execute cut-over
- Conduct “post-mortem” project analysis

RUP Best Practices

- Adapt the process
- Balance stakeholder priorities
- Collaborate across teams
- Demonstrate value iteratively
- Elevate the level of abstraction
- Focus continuously on quality

Extreme Programming (XP)[1]

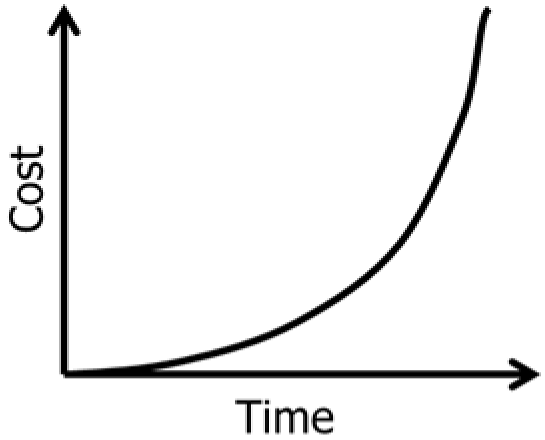
- A set of values, principles and practices for rapidly developing high-quality software that provides the highest value for the customer in the fastest way possible
- Take known good practices to extreme levels
- Most useful when plan is completely unclear

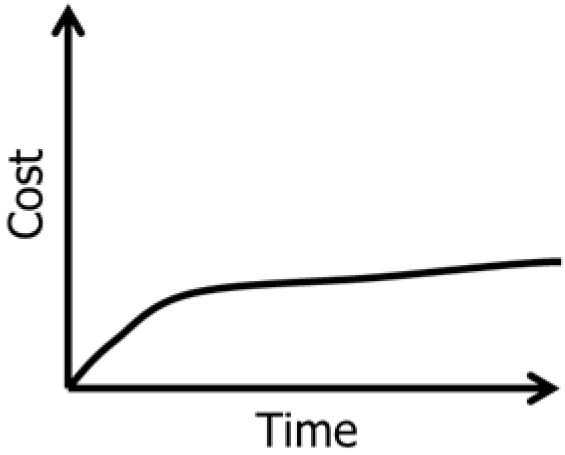
[1] Beck, Kent. "Embracing change with extreme programming." *Computer* 32.10 (1999): 70-77.

Premise of XP

- Traditional
  - Upfront design
  - Code late
  - Release when 'done'

- XP
  - Code early
  - Release early
  - Continuous design

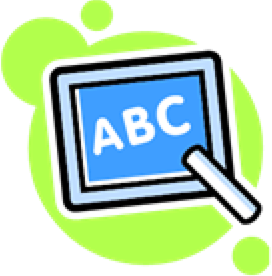


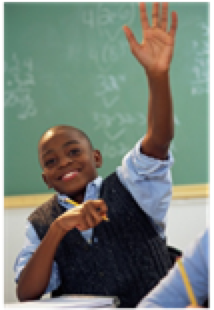


XP Five Values

1. Communication

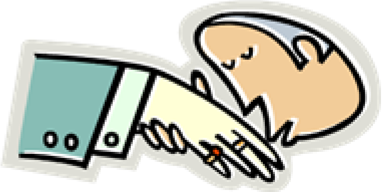
2. Simplicity





3. Feedback

4. Courage



5. Respect

XP Twelve Practices (1/2)

- Fine Scale Feedback
  - Pair Programming
  - Planning Game
  - Test Driven Development
  - Whole Team

- Continuous Process
  - Continuous Integration
  - Design Improvement
  - Small Releases

XP Twelve Practices (2/2)

- Shared Understanding
  - Coding Standards
  - Collective Code Ownership
  - Simple Design
  - System Metaphor

- Programmer Welfare
  - Sustainable Pace

XP Practice Example: Pair Programming (1/3)

- Brining code review practice to the extreme level!
  - Pair programming can be considered as real-time code review
  - We will apply this practice in our course project
- What’s good?
  - Resulting code has fewer defects
  - Programmers can interactively learn from each other
  - Enhances team-building and communication
- What’s bad?
  - Increased person-hours

Source: Padberg, Frank, and Matthias M. Muller. "Analyzing the cost and benefit of pair programming." *Proceedings. 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry (IEEE Cat. No. 03EX717)*. IEEE, 2004.

XP Practice Example: Pair Programming (2/3)

- Two programmers works together at one workstation

- **Driver**
  - Writes code

- **Navigator** (or **Observer**)
  - Reviews each line of code as it is typed in
  - Considers ‘strategic’ direction of the work, coming up with ideas for improvements and likely future problems to address

XP Practice Example: Pair Programming (3/3)

- Expert-Expert
  - Often yields new ways to solve problems, as both parties are unlikely to question established practices
- Expert-Novice: often preferred
  - Expert is now required to explain established practices, and prevent novice to just passively ‘watch the master’
  - Diverse ideas can pop up
- Novice-Novice
  - Significantly better than working independently, but also has disadvantages because it is hard to develop good habits

XP Work Products

- XP does not mean NO documentation
- XP focuses on code products
  - Automated test
  - System code
- Other products should be produced if
  - Necessary for the system
  - Helpful in producing code
- Typical non-code products
  - Open bug graph, task list and completion graph, risk list

XP Roles

- The Customer
  - Drives the project
- The Developer
  - Estimate his own work
  - Turn customer stories into working code
- The Coach
  - Guides and mentors the team
- The Tracker
  - Keep track of the schedule
  - Collect metrics

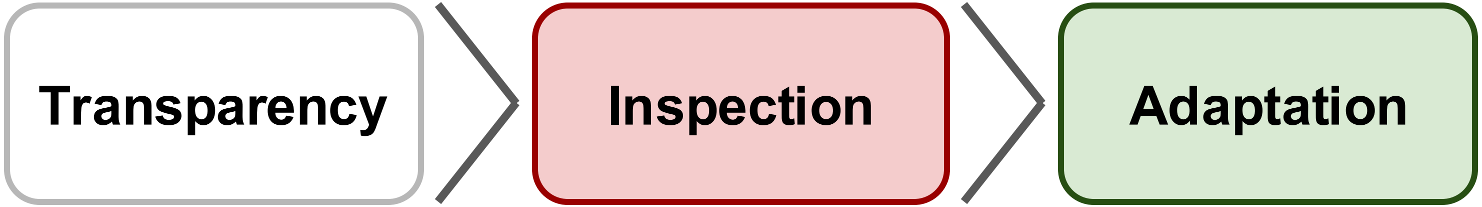
Scrum

- Scrum is a lightweight framework that helps people, teams and organizations generate value through adaptive solutions for complex problems.[1]
- The framework itself is adjusted to follow the trends
  - Last updated in 2020
- “Sprint”
  - A unit cycle (1~4 weeks of rigidly fixed length) of iteration
  - One after each other
  - A sprint goal consists of a **working product**

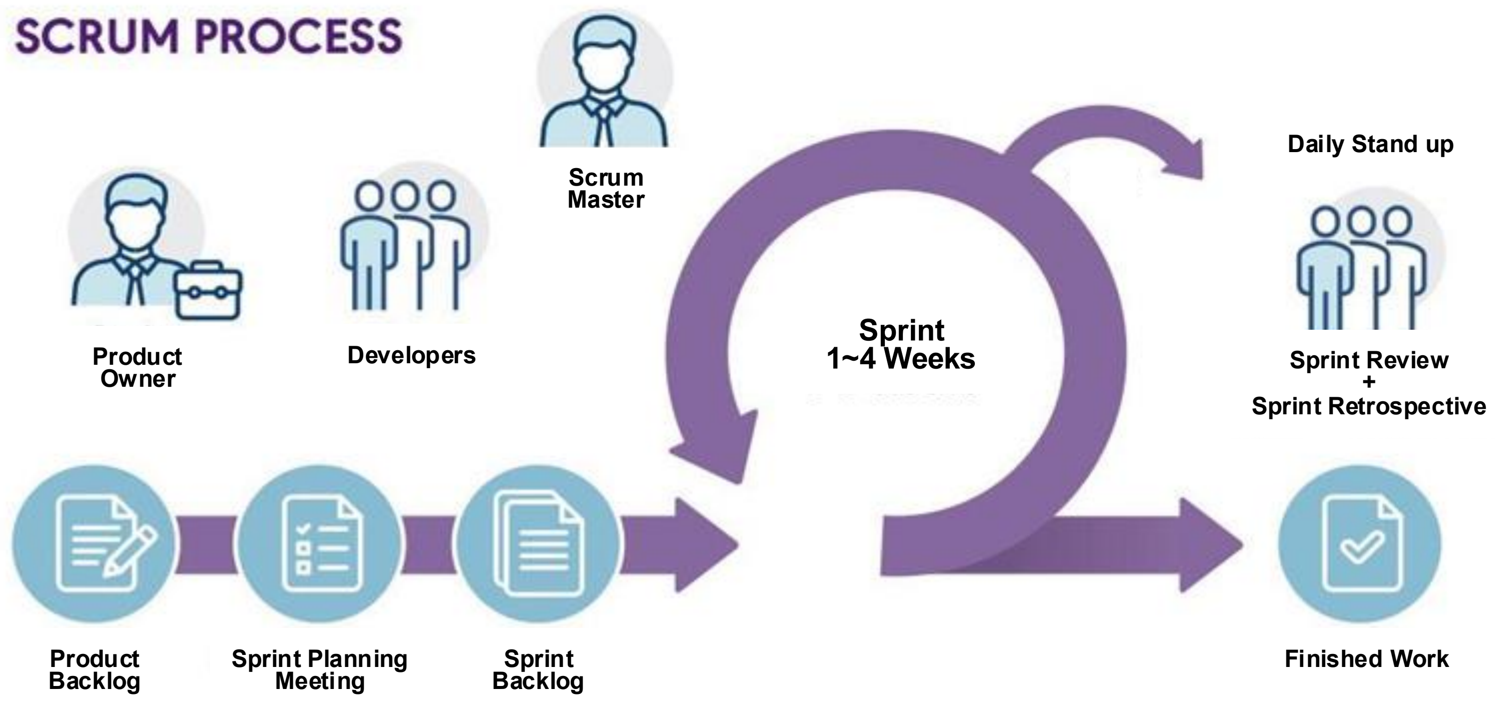
[1] Schwaber, Ken, and Jeff Sutherland. "The Scrum Guide. 2020."

Scrum: Three Goals

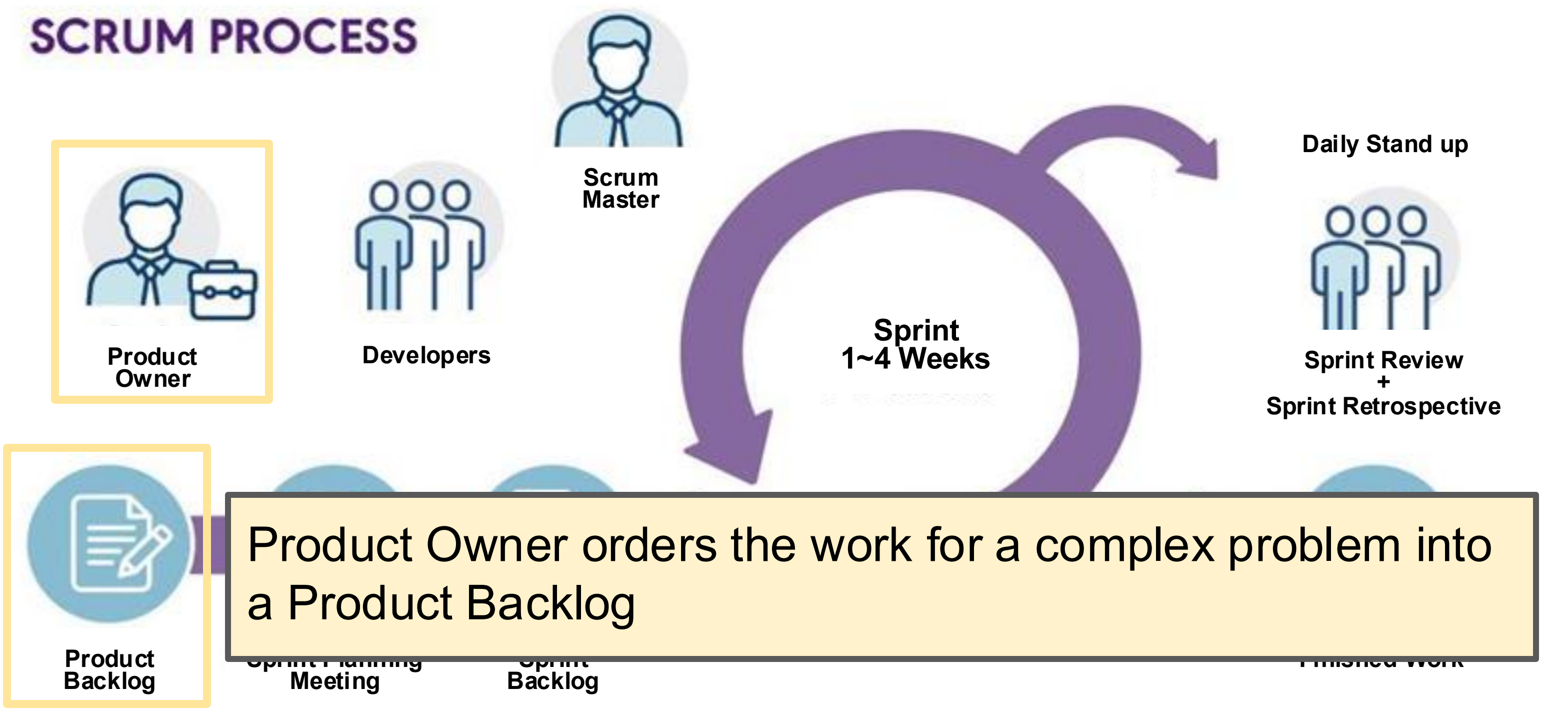
- Transparency
  - Artifacts with low transparency can lead to decisions that diminish value and increase risk.
- Inspection
  - Frequent and diligent inspection to detect undesirable variances
- Adaptation
  - The ultimate goal



Scrum

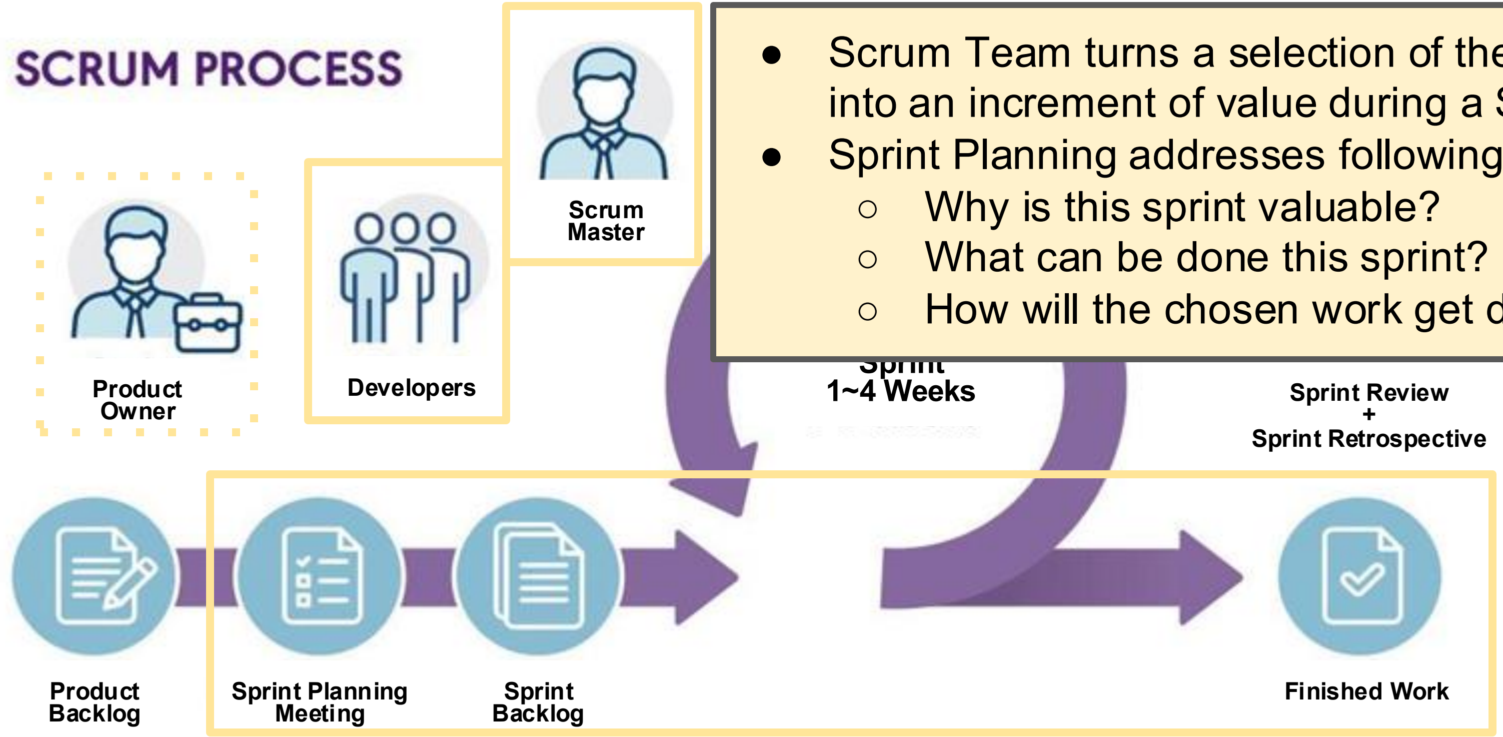

SCRUM PROCESS
Scrum Master
Product Owner
Developers
Daily Stand up
Sprint
1~4 Weeks
Sprint Review
+
Sprint Retrospective
Product Backlog
Sprint Planning Meeting
Sprint Backlog
Finished Work

Scrum: Basic Procedure



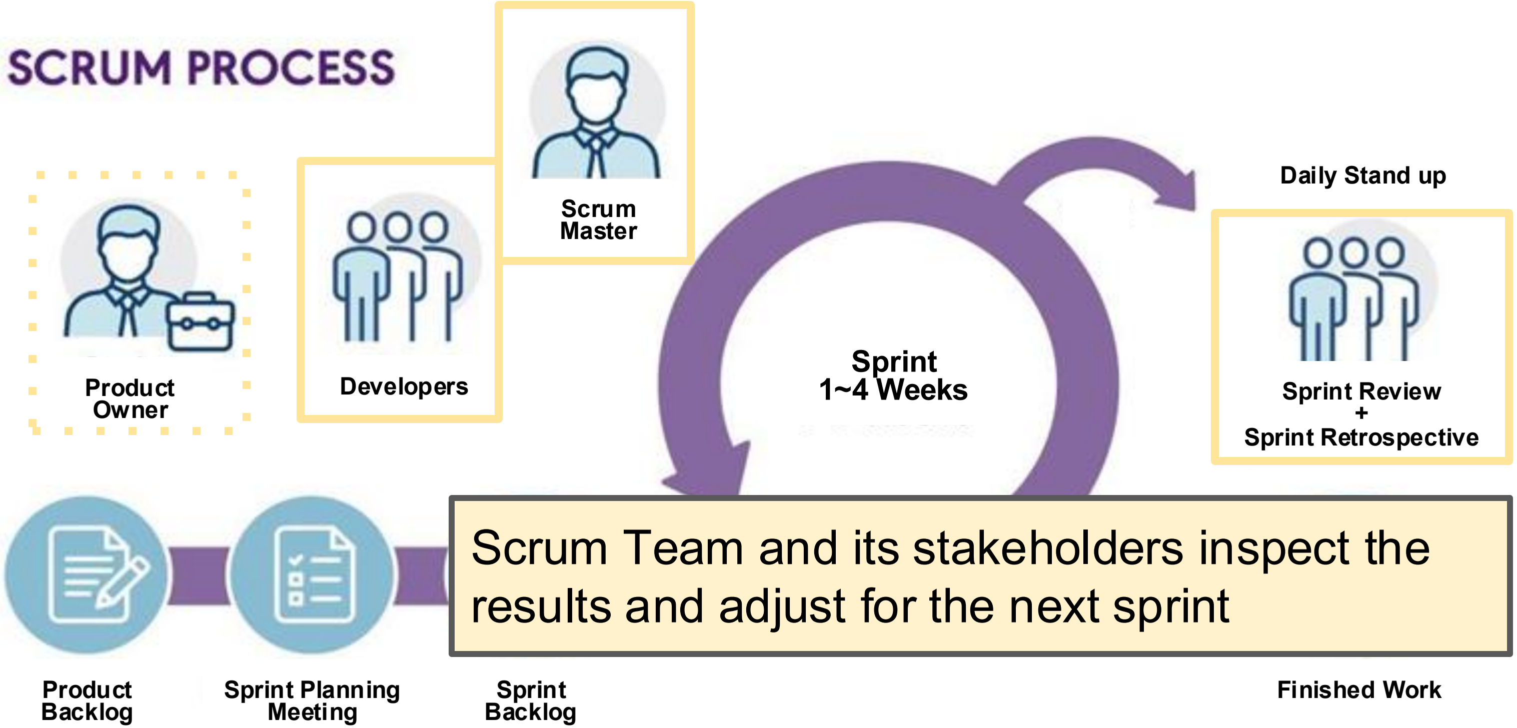
Product Owner orders the work for a complex problem into a Product Backlog

Scrum: Basic Procedure



- Scrum Team turns a selection of the work into an increment of value during a Sprint
- Sprint Planning addresses following topics
  - Why is this sprint valuable?
  - What can be done this sprint?
  - How will the chosen work get done?

Scrum: Basic Procedure



Scrum Team and its stakeholders inspect the results and adjust for the next sprint

Scrum: Daily Scrum

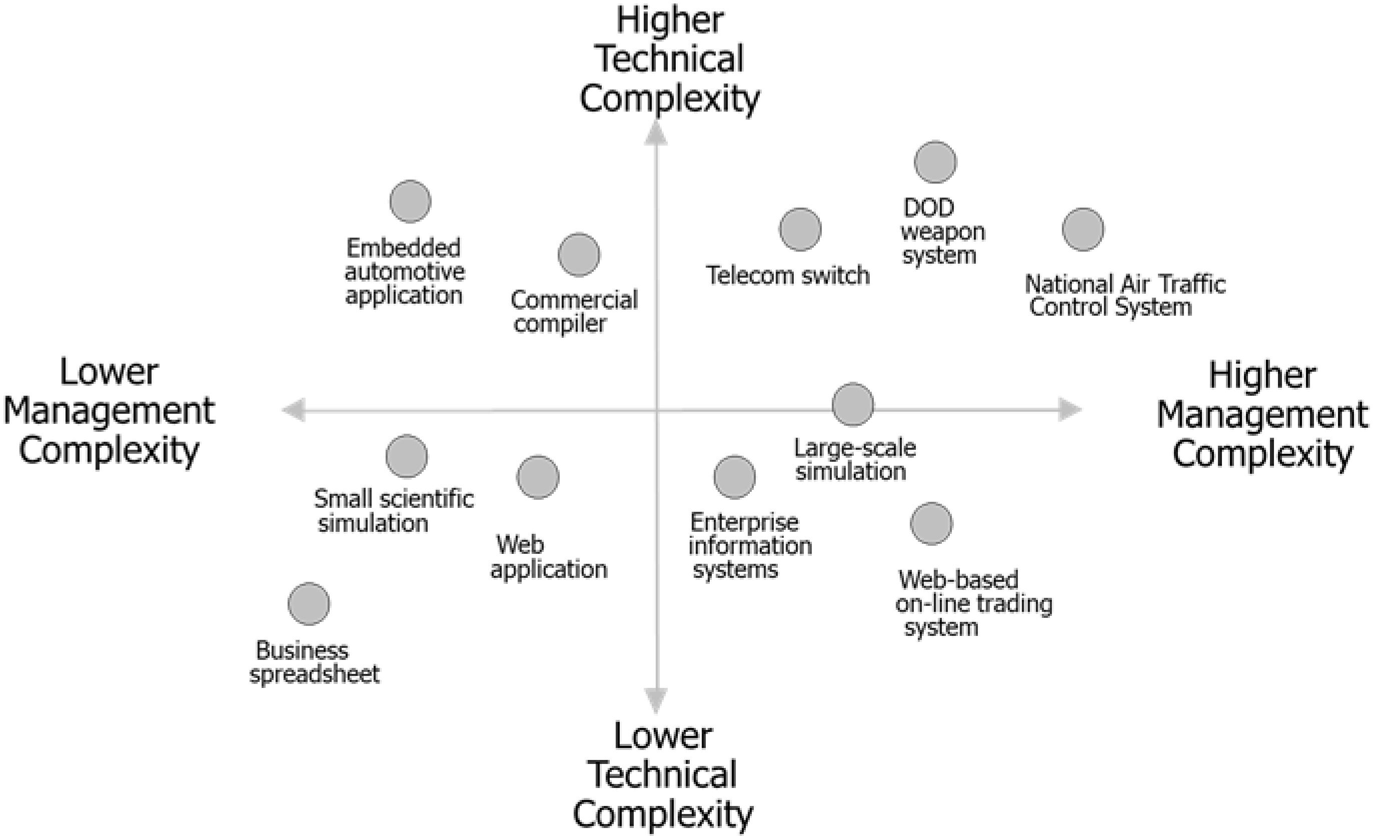
- Daily Scrum is a 15-minute event for the Developers of the Scrum Team held at the same time and place every working day of the Sprint.
- Developers can select whatever structure and techniques they want.
- Daily Scrums improve communications, identify impediments, promote quick decision-making, and consequently eliminate the need for other meetings.
- Daily Scrum is not the only time Developers are allowed to adjust their plan.

Daily Stand up

Sprint Review
+
Sprint Retrospective

Finished Work

Can a Single Process Fit All?

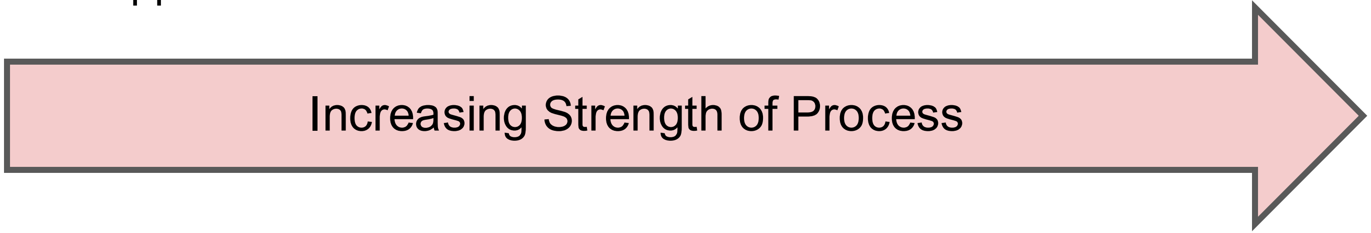

Higher Technical Complexity
Embedded automotive application
Commercial compiler
Telecom switch
DOD weapon system
National Air Traffic Control System
Lower Management Complexity
Higher Management Complexity
Large-scale simulation
Small scientific simulation
Web application
Enterprise information systems
Web-based on-line trading system
Business spreadsheet
Lower Technical Complexity

How Much Process is Necessary?

- Simple upgrades
- R&D prototypes
- Static web applications

- Component based (.NET, JEE)
- Dynamic web applications

- Legacy upgrades
- Real-time systems



Increasing Strength of Process

- Co-located teams
- Few stakeholders
- Smaller, simpler projects

- Distributed teams
- Many stakeholders
- Large projects
- Lots of uncertainty

Choosing Appropriate Process

- Consider 5 aspects
    - Alignment with project goals
    - Efficiency and productivity
    - Quality control
    - Adaptability and flexibility
    - Risk mitigation and quality assurance

Summary

- Systems Development Life Cycle (SDLC) is phase-divided description of a development process
- There are various software process models, each highlighting different values
- Waterfall model is a rigid sequential development process
- Agile methods are based on iterative & incremental development with rapid flexible response to change