Module 40: Software Engineering

Software Development Life Cycle (SDLC)

Instructors: Abir Das and Jibesh Patra
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

{*abir, jibesh*}*@cse.iitkgp.ac.in*

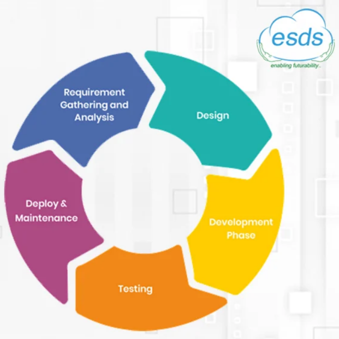
Slides taken from NPTEL course on Programming in Modern C++

by **Prof. Partha Pratim Das**

Table of Contents

Software Development Life Cycle (SDLC)

- **SDLC** stands for *Software Development Life Cycle*
- A process for developing, designing, and maintaining a software project by ensuring that all the functionalities along with user requirements, objectives, and end goals are addressed
- With SDLC, the software project's quality and the overall software development process get enhanced.



Goals of SDLC

Organizational objectives are:

- **Efficiency**
  - Build the system RIGHT: Do things RIGHT
    - **Windows 98: Crashed on launch**
    - **Windows 10: Most popular OS**
  - Lines of *quality* code produced per man-hour
  - Instances of *quality* support provided per man-week
  - **Minimize Rework**: *Get it right the first time*
- **Effectiveness**
  - Build the RIGHT system: Do RIGHT things
    - **Iridium**
    - **Tesla**
  - Lines of *quality* code produced per dollar
  - Cost for every instance of support
  - **Minimize Rework**: *Get it right the first time*

Processes ensure Efficiency & Effectiveness

Goals of SDLC

SDLC is about Process compliance

SDLC has three primary objectives

[1] Ensure that **high quality systems** are delivered

- Built to contract (commitment)

[2] Provide **strong management controls** over the projects

- Efficiency
- Effectiveness
- Process Compliance
- Cost Control
- Customer Satisfaction

[3] **Maximize the productivity** of the systems staff

- Built by best practices

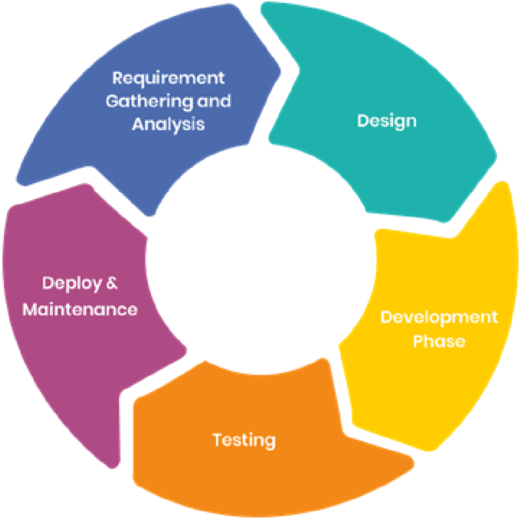
Better Quality at Lower Cost

Benefits of SDLC

SDLC offers the following benefits

- Address the goals and problems so that the project is implemented successfully
- Project members cannot proceed ahead before completion & approval of the prior stages
- Has necessary checks at each stage so that it is tested with precision before entering the installation stage
- Project members can continue the software development process without incurring any complications
- Optimal control with minimum problems, allowing the project members to run the project smoothly

Stages of SDLC



SDLC Stage 1: Requirement Gathering & Analysis Phase

Communication between

- stakeholders
- end-users
- project teams

for requirements are gathered from customers

- functional
- non-functional

This Phase of SDLC involves:

- Analysis of functionality and financial feasibility
- Identifying and capturing requirements of stakeholders through customer interactions like interviews, surveys, etc.
- Documenting all product requirements in a *Software Requirements Specification* (SRS) from customer
- Creating project prototypes

SDLC Stage 2: Design Phase

Architectural design is proposed based on the SRS Document requirements and its refinements

This Phase of SDLC involves:

- Separation of requirements
  - hardware and software system
  - functional and non-functional
- Designing the system architecture based on gathered requirements
- Creating *Unified Modelling Language* (UML) diagrams like use cases, class diagrams, sequence diagrams, and activity diagrams
- Creating
  - *High Level Design* (HLD)
  - *Low Level Design* (LLD)

SDLC Stage 3: Development Phase

Codes are written, and system is developed and built.

This Phase of SDLC involves:

- Actual code is written
- Demonstration of accomplished work presented before a Business Analyst for further modification of work
- Unit testing is performed, that is, verifying the code based on requirements

SDLC Stage 4: Testing Phase

- Almost all stages of SDLC involves the testing strategy
- SDLC's testing phase refers to checking, reporting, and fixing the system for any bug/defect
- The on-going project is migrated to a test environment where different testing forms are performed
- Continues until the project has achieved the quality standards, as mentioned in the SRS

This Phase of SDLC involves:

- Testing the system as a whole
- Performing different types of test in the system
- Reporting and fixing all forms of bugs & defects

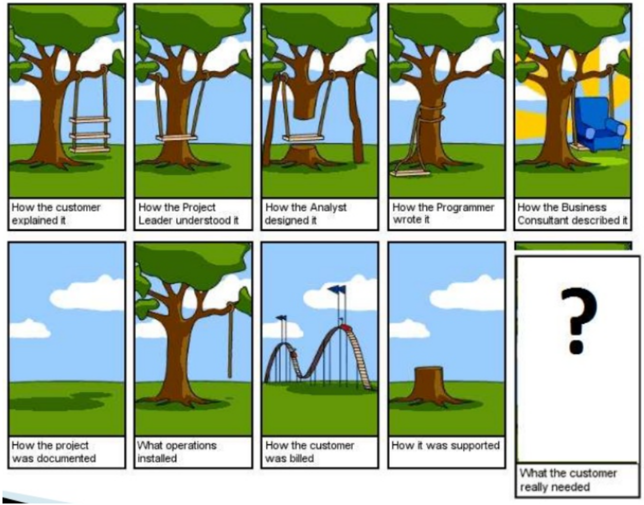
SDLC Stage 5: Deployment & Maintenance Phase

- Ready to be launched
- May be initially released for limited users by testing it in a real business environment for *User Acceptance Testing* (UAT)

This Phase of SDLC involves:

- The system is ready for delivery
- The system is installed and used
- Errors are rectified that might have been previously missed
- Enhancing the system inside a data center

How Software Projects Fail


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
?
What the customer really needed

Source: *Introduction to SCRUM*

SDLC Models

Various SDLC models are defined and designed to follow the software development process. These models are also known as Software Development Process Models. Each of these models follows a series of steps for ensuring the complete success of a project.

Some of the most popular SDLC models used for software development include:

[1] Waterfall Model

[2] Iterative-Incremental Model

[3] V Shaped Model

[4] Rapid Action Development (RAD) Model

[5] Spiral Model

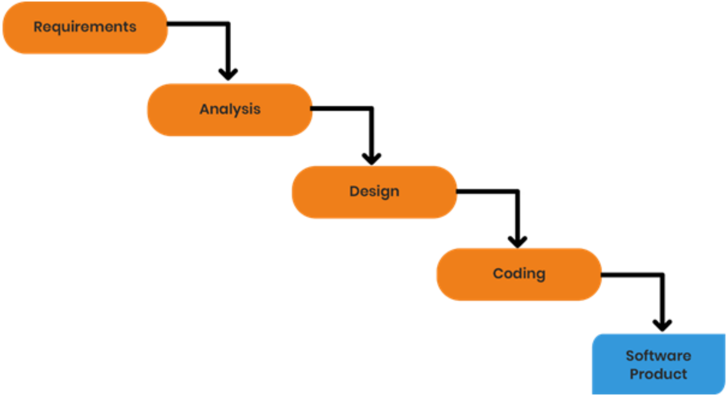
[6] Agile Model

SDLC Model Timelines

Decade	**Methodology**
1950s	Code & Fix
1960s	Design-Code-Test-Maintain
1970s	Waterfall Model
1980s	Spiral Model
1990s	Rapid Application Development, V Model
2000s	Agile Methods

Waterfall Model

This model is the most commonly used SDLC model. In this model, each phase starts only after the previous step has been completed. This is a linear model having no feedback loops.



Waterfall Model

- Strengths of the Waterfall Model
  - Easy to understand and use
  - Achievements are well-defined
  - Defines requirements stability
  - Works well when the project quality is important
- Weaknesses of the Waterfall Model
  - It cannot match reality well
  - Difficult to make changes
  - Software delivered towards the end of the project only
  - Testing begins only after the development phase is complete

Customer at the START and END only

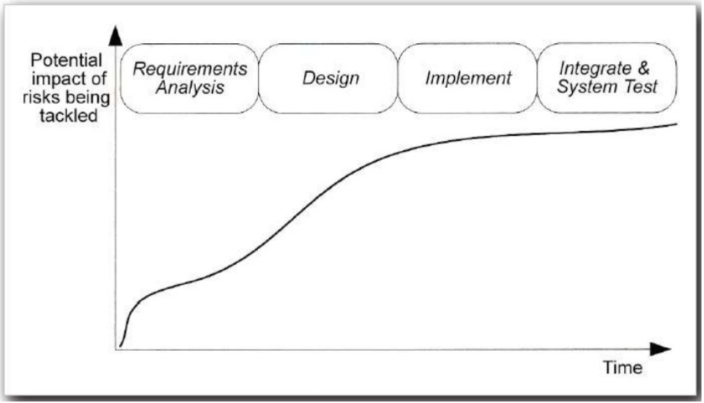
RIGHT recipe to build a WRONG system

Waterfall Model

Works when:

- Software projects that are stable
- Unchanging requirements
- Where it is possible that the designers will be able to fully predict problems
- Where it is possible that the designers produce a *correct* design

Waterfall Model: Fail-late Lifecycle

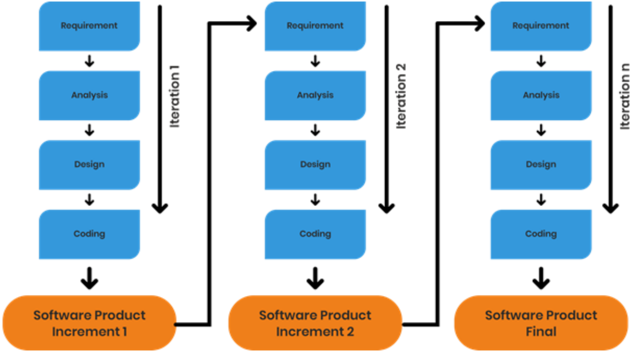


Source: *Introduction to SCRUM*

- Bug found early is cheaper in money, effort & time to fix than same bug found later on
- An early defect that is left undetected until development or maintenance is estimated to cost 50 to 200 times as much to fix as it would have cost to fix at requirements time

Iterative-Incremental Model

- In this model, in the initial stages, a partial implementation of the complete system is constructed such that it will be present in a deliverable form.
- Increased functionalities are added and for any defects, they are fixed with the working product delivered at the end.
- This process is repeated until the product development cycle gets completed.
- These repetitions of processes are known as **iterations**. With each iteration, a product increment gets delivered.



Iterative-Incremental Model

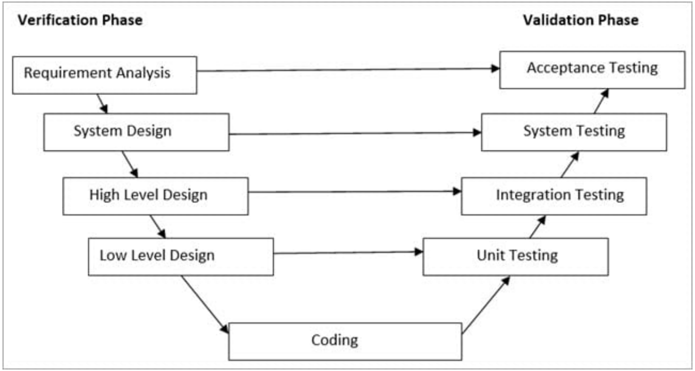
- Strengths of the Iterative-Incremental Model
  - Prioritized requirements can be initially developed
  - The initial delivery of the product is faster
  - Lower initial delivery costs
  - Changes in requirements can be easily adjusted
- Weaknesses of the Iterative-Incremental Model
  - There are requirements for effective iterations planning
  - Efficient design is required for including the required functionalities
  - An early definition of a complete, as well as fully functional system, is needed for allowing increments definition
  - Clear module interfaces are required

Customer at the START and END of Iterations only

Better than Waterfall

V-Shaped Model

V- Model is also known as **Verification** and **Validation** Model. In this model Verification & Validation goes hand in hand, that is, development and testing goes parallel. V model and Waterfall model are the same except that the test planning and testing start at an early stage in V-Model.



Source: *SDLC (Software Development Life Cycle) Phases, Methodologies, Process, And Models*

V-Shaped Model

- Strengths of the V-Shaped Model
  - It is a simple and easily understandable model
  - V–model approach is good for smaller projects wherein the requirement is defined and it freezes in the early stage
  - It is a systematic and disciplined model which results in a high-quality product
- Weaknesses of the V-Shaped Model
  - V-shaped model is not good for ongoing projects
  - Requirement change at the later stage would cost too high

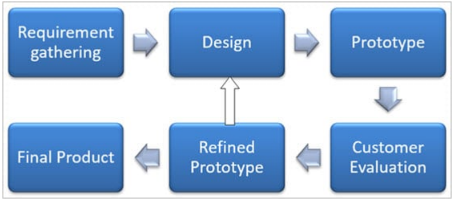
Customer at the START and END only

Works well for small & new projects

Source: *SDLC (Software Development Life Cycle) Phases, Methodologies, Process, And Models*

Rapid Application Development (RAD) Model: Prototype Model

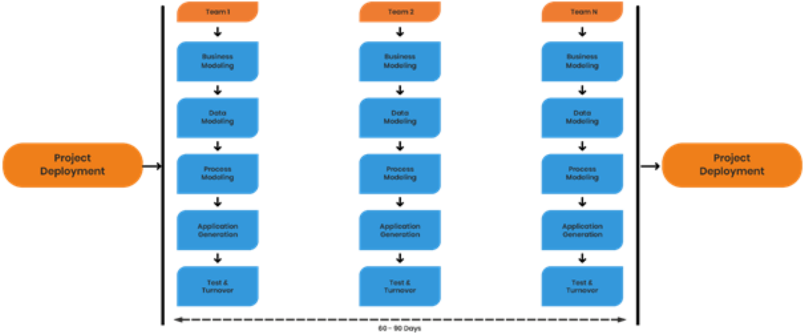
- RAD is a refinement of the **Prototype** model in which the prototype is developed prior to the actual software
- Prototype models have limited functional capabilities and inefficient performance when compared to the actual software.
- Dummy functions are used to create prototypes
- Prototypes are built prior to the actual software to get valuable feedback from the customer
- Feedbacks are implemented and the prototype is again reviewed by the customer for any change
- This process goes on until the model is accepted by the customer.



Source: *SDLC (Software Development Life Cycle) Phases, Methodologies, Process, And Models*

Rapid Application Development (RAD) Model

The RAD SDLC model is based on **prototyping** and **iterative development**, with no involvement of a defined planning structure. In this model, different function modules are parallelly developed as prototypes and then integrated to speed up product delivery.



Rapid Application Development (RAD) Model

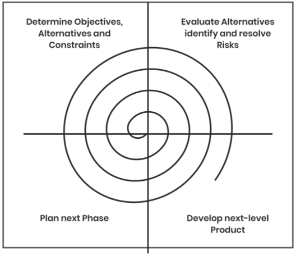
- Strengths of the RAD Model
  - Reduced cycle time and enhanced productivity with minimal team members
  - Customer's continuous involvement ensures minimal risks of not achieving customer satisfaction
  - Easy to accommodate any user changes
- Weaknesses of the RAD Model
  - Hard to use and implement with legacy systems
  - Heavily dependent on technically strong members for identifying business requirements

Customer at the Prototyping

Works well for fast development priorities
(Smaller development)
(Development for estimation)

Spiral Model

The spiral model combines **risk analysis** along with **RAD prototyping** to the **Waterfall model**. The loops in the model represent the phase of the SDLC process – the innermost loop is of requirement gathering & analysis which follows the Planning, Risk analysis, development, and evaluation. Next loop is Designing followed by Implementation & then testing.


Determine Objectives, Alternatives and Constraints
Evaluate Alternatives identify and resolve Risks
Plan next Phase
Develop next-level Product

The spiral model has 4 quadrants:

[1] Determine Objectives, Alternatives and Constraints (Quad 1: *Planning*)

[2] Evaluate Alternatives, Identify and Resolve Risks (Quad 2: *Risk Analysis*)

[3] Develop Next-Level Product (Quad 3: *Engineering*)

[4] Planning the Next Phase (Quad 4: *Evaluation*)

Spiral Model

- Strengths of the Spiral Model
  - An early indication of the risks can be provided, without incurring much cost
  - Users can have a look at their system early due to RAD
  - Users are involved in all lifecycle stages
  - Critical & high-risk functionalities are initially developed
- Weaknesses of the Spiral
  - Hard to set the objectives, verifiable milestones for indicating preparedness to go ahead with the next iteration
  - Time spent on addressing risks can be large for smaller & low-risk involved projects
  - Complex to understand for new members
  - The spiral may go on indefinitely

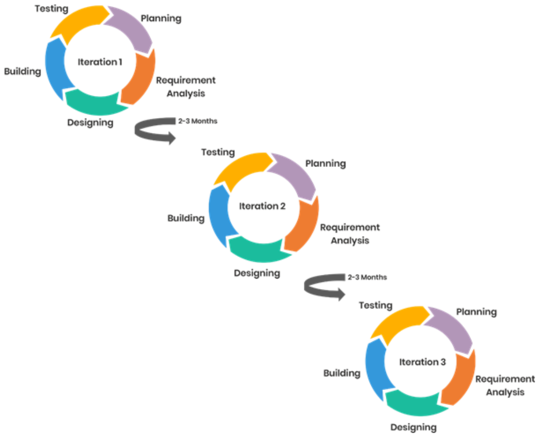
Customer at every SPIRAL round

Improves Quality, Reduces Rework

May slow down development

Agile Model

The agile model is the combination of the **iterative-incremental model** that depends on process adaptability along with **customer satisfaction** through the delivery of software products. In this model, the project is broken down into smaller time frames for delivering certain features during a release.



Agile Model

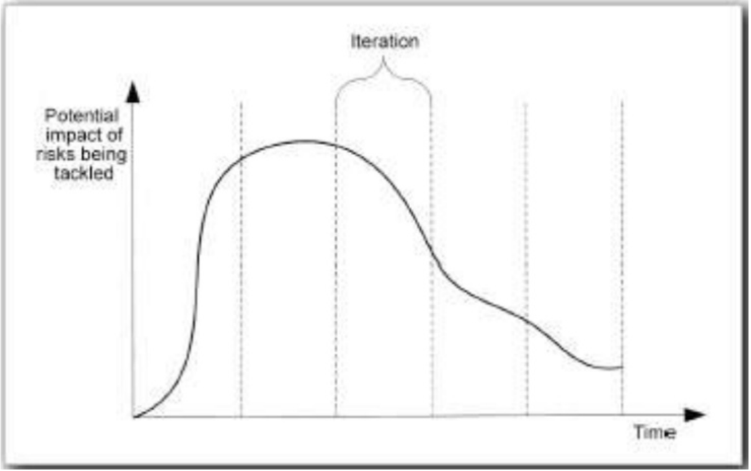
- Strengths of the Agile Model
  - Easy to accommodate changing requirements
  - Regular communication takes place between customers and developers
  - Functionalities can be developed quickly and demonstrated to customers
- Weaknesses of the Agile Model
  - Not ideal for handling complex dependencies
  - Teams need to have the desired experience levels for adhering method rules

Customer in the LOOP

Improves Quality, Reduces Rework

May slow down development

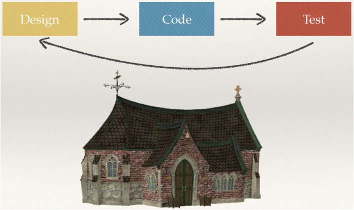
Agile Approach: Fail-early Lifecycle



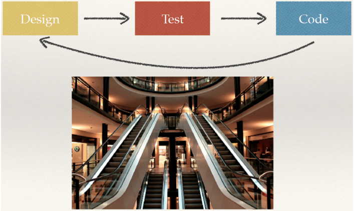
Source: *Introduction to SCRUM*

Test Driven Development

Old School



New School



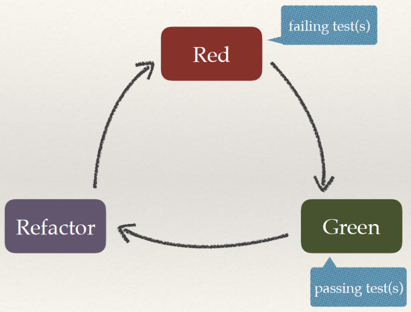
Source: *An Introduction to Test Driven Development*

Test Driven Development

- **Test-Driven Development** (TDD) is a technique for building software that guides software development by writing tests. (Martin Fowler's definition)
- TDD is NOT primarily about testing or development (that is, coding)
- It is rather about design - where design is evolved through refactoring
  - Developers write unit tests (NOT testers) and then code
- In TDD, tests mean *Unit Tests*
- When unit tests are written in a project, TDD may not be followed - tests could be written after the writing code: *Plain Old Unit testing* (POUting)
  - *Following TDD, we write the tests first* **before** *writing the code*
- **Disadvantages** *in writing tests* **after** *code*
  - Testing does not give direct f/b to design and programming ⇒ in TDD, the f/b is directly fed back into improving design & programs
  - Often, after realising the functionality in code, unit testing is omitted ⇒ TDD inverts this sequence and helps create unit tests first
  - Writing tests after developing code often results in **happy path** testing ⇒ we don't have enough granular or **testable** code segments to write the tests

Source: *An Introduction to Test Driven Development*

TDD Mantra



- **Red**: Write a little test that doesn't work, perhaps doesn't even compile at first
- **Green**: Make the test work quickly, committing whatever sins necessary in the process
- **Refactor**: Eliminate all the duplication and smells created in just getting the test to work

Source: *An Introduction to Test Driven Development*

TDD Laws

Best Practice

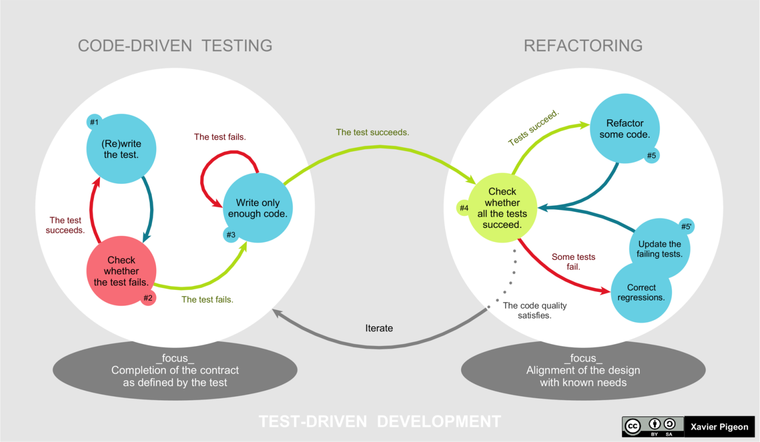
Make it green, then make it clean!

Three Laws of TDD:

- **Law 1**: You may not write production code unless you've first written a failing unit test.
- **Law 2**: You may not write more of a unit test than is sufficient to fail.
- **Law 3**: You may not write more production code than is sufficient to make the failing unit test pass.

Source: *An Introduction to Test Driven Development*

TDD Global Lifecycle



Source: *Wikipedia::Test-driven development*

TDD Benefits

Benefit	Reason
Better Design	Cleaner code (because of refactoring)
Safer refactoring	Increased quality
Better code coverage	Tests serve as documentation
Faster debugging	Most often, the failing code / test is in the most recently changed code
Self-documenting tests	Test-cases show / indicate how to use the code

Source: *An Introduction to Test Driven Development*

Agile Methods

- Agile Modeling
- Agile Unified Process (AUP)
- Dynamic System Development Method (DSDM)
- Essential Unified Process (EssUP)
- **Extreme Programming (XP)**
- Feature Driven Development (FDD)
- Open Unified Process (OpenUP)
- **Scrum**
- Velocity Tracking

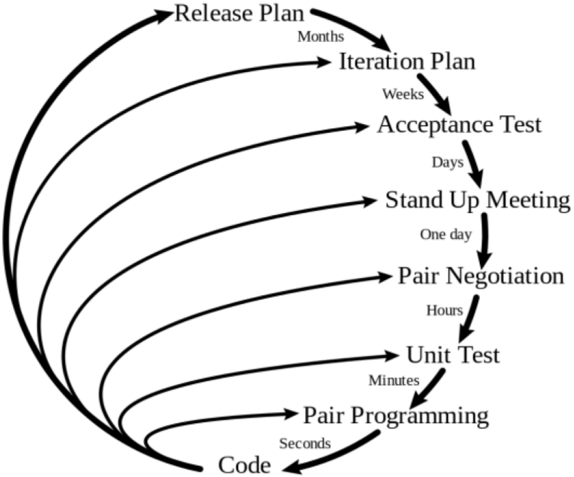
Source: *Introduction to SCRUM*

Extreme Programming: Overview

- XP is an Agile Model
- Created by Kent Beck during his work on the C3 project when he became the project leader in 1996
- Kent wrote a book on the methodology: *Extreme Programming Explained: Embrace Change* (October 1999)
- Kent is a leading proponent of TDD

Source: *The Extreme Programming (XP) Model*

Extreme Programming: Planning / Feedback Loop



Source: *The Extreme Programming (XP) Model*

Extreme Programming

- Start with the **Planning Game**:
- The *game is a meeting* that:
  - Occurs once per iteration
  - Typically once a week
- The *planning process* is divided into two parts:
  - *Release Planning*:
    - This is focused on determining what requirements are included in which near-term releases, and when they should be delivered.
    - The customers and developers are both part of this
  - *Iteration Planning*:
    - This plans the activities and tasks of the developers.
    - In this process the customer is not involved.

Source: *The Extreme Programming (XP) Model*

Extreme Programming: Practices (12): Management-Practices (3)

- **Management-Practices**
  - ○ *On-Site Customer*
    - ▷ A central customer contact must always be accessible in order to clarify requirements and questions directly
  - ○ *Planning Game*
    - ▷ Projects, in accordance with XP, run iteratively (repeatedly) and incrementally (gradually build on each other)
    - ▷ The contents of the next step are planned before each iteration
    - ▷ All project members (including the customer) participate
  - ○ *Short Releases*
    - ▷ New deliveries should be made at short intervals
    - ▷ Customers receive the required functions quicker and can therefore give feedback on the development quicker

Source: *The Extreme Programming (XP) Model*

Extreme Programming: Practices (12): Team-Practices (5)

- **Team-Practices**
  - *Metaphor*
    - ▷ Only a few clear metaphors should describe the system for better clarity
  - *Collective Ownership*
    - ▷ The whole team is responsible for the system, not individuals
    - ▷ Each developer must have access to all lines of code
    - ▷ Each developer is able to take over the task of another developer
  - *Continuous Integration*
    - ▷ All changes to the system are integrated promptly so that not too many dependencies between changes occur
  - *Coding Standards*
    - ▷ There should be a common standard for writing the code
  - *Sustainable Pace*
    - ▷ XP builds on the creativity of the individual project members
    - ▷ Creativity cannot be achieved by constantly working overtime
    - ▷ Overtime is to be avoided

Source: *The Extreme Programming (XP) Model*

Extreme Programming: Practices (12): Programming-Practices (4)

- **Programming-Practices**
  - *Testing*
    - All developments must be tested
    - TDD is preferred
  - *Simple Design*
  - *Refactoring*
    - As soon as it becomes necessary to alter the structure of the system, it should be implemented
    - Needed for TDD as well
  - *Pair Programming*
    - Two programmers work together at one workstation
    - One, the **driver**, writes code while the other, the **observer** or **navigator**, reviews each line of code as it is typed in.
    - The two programmers switch roles frequently
    - Observer also considers the *strategic* direction of the work, ideas for improvements and likely future problems to address
    - Driver focuses on the *tactical* aspects of completing the current task, using the observer as a safety net and guide

Source: *The Extreme Programming (XP) Model*

Extreme Programming

- Strengths of XP
  - Large project are divided into manageable amounts
  - Reduced costs and time required for project realization
  - XP teams saves money because they don't use limited documentation
  - Simplicity is another advantage of XP projects
  - Simplicity of XP leads to faster completion with less defects
  - XP is reduces the risks related to programming – using module structure, and pair programming to spreads the risk and mitigate the dependence on individuals
  - TDD at the coding stage and the customer UAT validation leads to successful development completion
- Weaknesses of XP
  - XP is focused on the code rather than on design
  - XP requires a detailed planning from the start due to changing costs & scope
  - XP doesn't measure/plan Quality Assurance of coding
  - Developers' comfort is low, requires more discipline in the team and devotion of customers
  - Project management might experience difficulties related with the practice that changes during the life cycle
  - XP is practiced with pair programming which might usually lead to too much duplication of codes and data

Source: *The Extreme Programming (XP) Model*

SCRUM

Sweep

Waterfall

Reverse Sweep

SCRUM

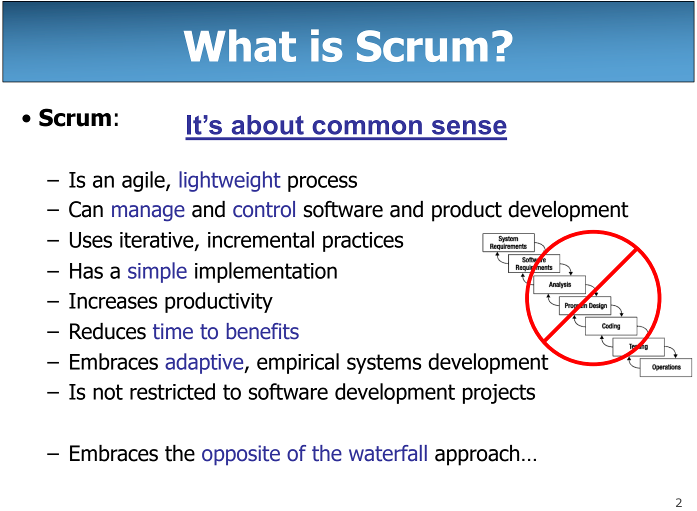
SCRUM

A **scrum** (short for *scrummage*) is a method of restarting play in rugby football that involves players packing closely together with their heads down and attempting to gain possession of the ball. ... Both teams may then try to compete for the ball by trying to hook the ball backwards with their feet.

Discussion on SCRUM SDLC is lifted from:

Source: *CSE 403 Lecture Slides - Washington*

SCRUM



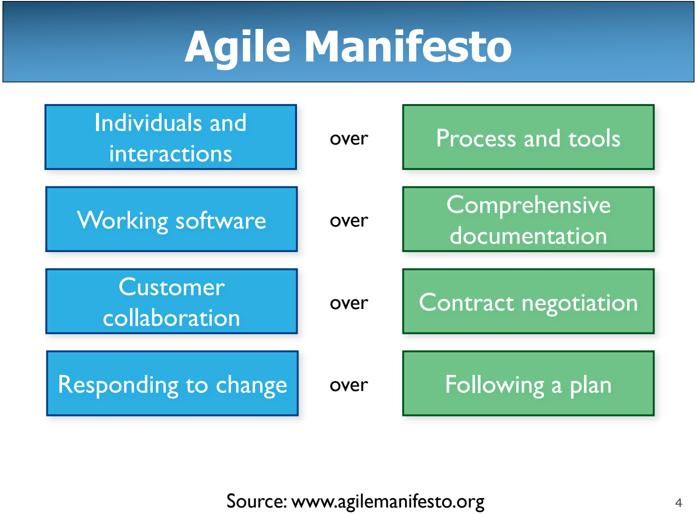
What is Scrum?

- **Scrum**: It’s about common sense
  - Is an agile, lightweight process
  - Can manage and control software and product development
  - Uses iterative, incremental practices
  - Has a simple implementation
  - Increases productivity
  - Reduces time to benefits
  - Embraces adaptive, empirical systems development
  - Is not restricted to software development projects
  - Embraces the opposite of the waterfall approach…

2

Source: *CSE 403 Lecture Slides - Washington*

SCRUM



Agile Manifesto

Individuals and interactions over Process and tools

Working software over Comprehensive documentation

Customer collaboration over Contract negotiation

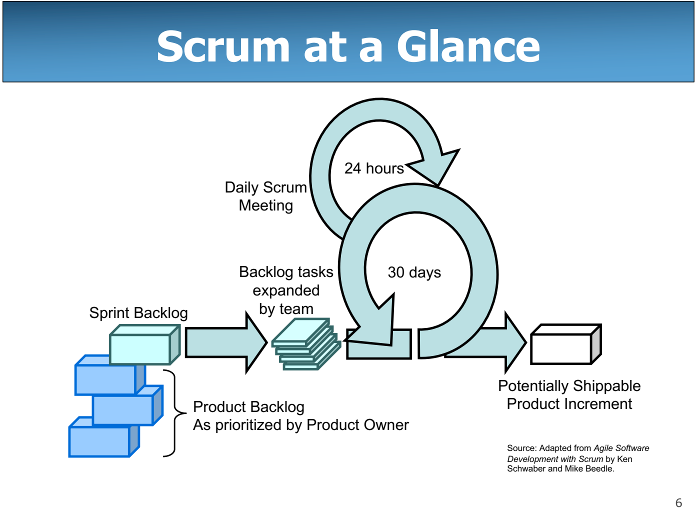
Responding to change over Following a plan

Source: www.agilemanifesto.org

4

Source: *CSE 403 Lecture Slides - Washington*

SCRUM



Scrum at a Glance

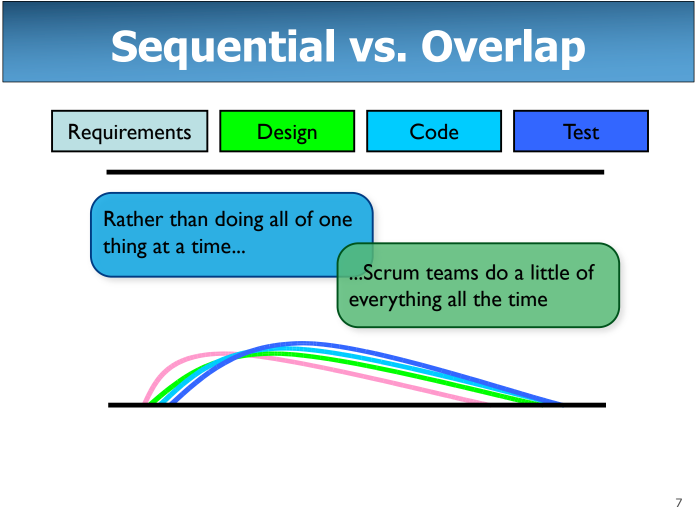
24 hours
Daily Scrum Meeting
30 days
Backlog tasks expanded by team
Sprint Backlog
Product Backlog
As prioritized by Product Owner
Potentially Shippable Product Increment

Source: Adapted from *Agile Software Development with Scrum* by Ken Schwaber and Mike Beedle.

6

Source: *CSE 403 Lecture Slides - Washington*

SCRUM



Sequential vs. Overlap

Requirements | Design | Code | Test

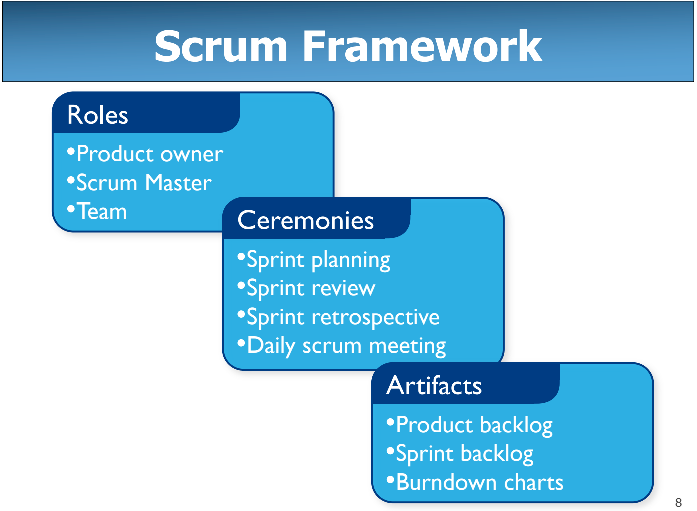
Rather than doing all of one thing at a time...

...Scrum teams do a little of everything all the time

7

Source: *CSE 403 Lecture Slides - Washington*

SCRUM



Scrum Framework

Roles

- Product owner
- Scrum Master
- Team

Ceremonies

- Sprint planning
- Sprint review
- Sprint retrospective
- Daily scrum meeting

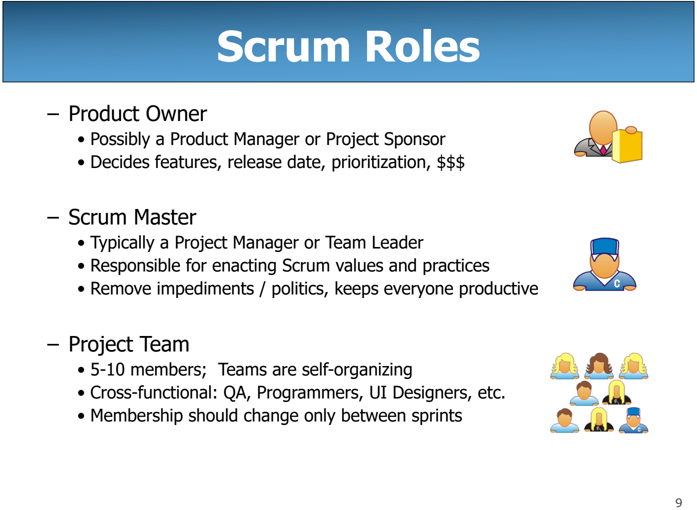
Artifacts

- Product backlog
- Sprint backlog
- Burndown charts

8

Source: *CSE 403 Lecture Slides - Washington*

SCRUM



Scrum Roles

– Product Owner
  - Possibly a Product Manager or Project Sponsor
  - Decides features, release date, prioritization, $$$

– Scrum Master
  - Typically a Project Manager or Team Leader
  - Responsible for enacting Scrum values and practices
  - Remove impediments / politics, keeps everyone productive

– Project Team
  - 5-10 members; Teams are self-organizing
  - Cross-functional: QA, Programmers, UI Designers, etc.
  - Membership should change only between sprints

9

Source: *CSE 403 Lecture Slides - Washington*