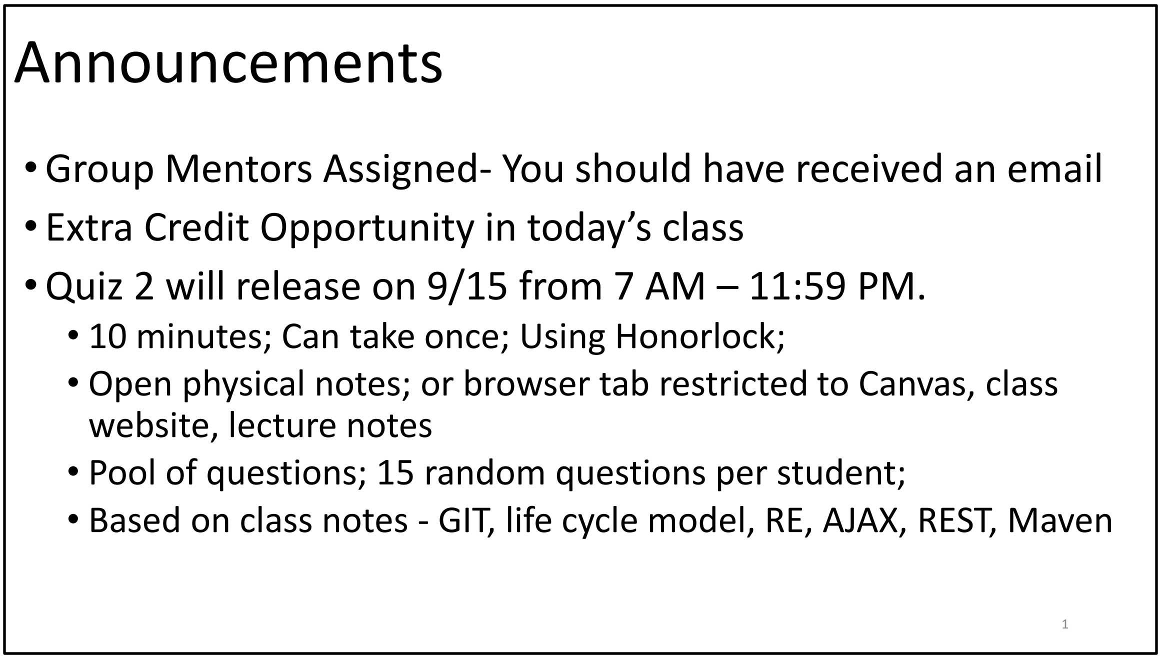
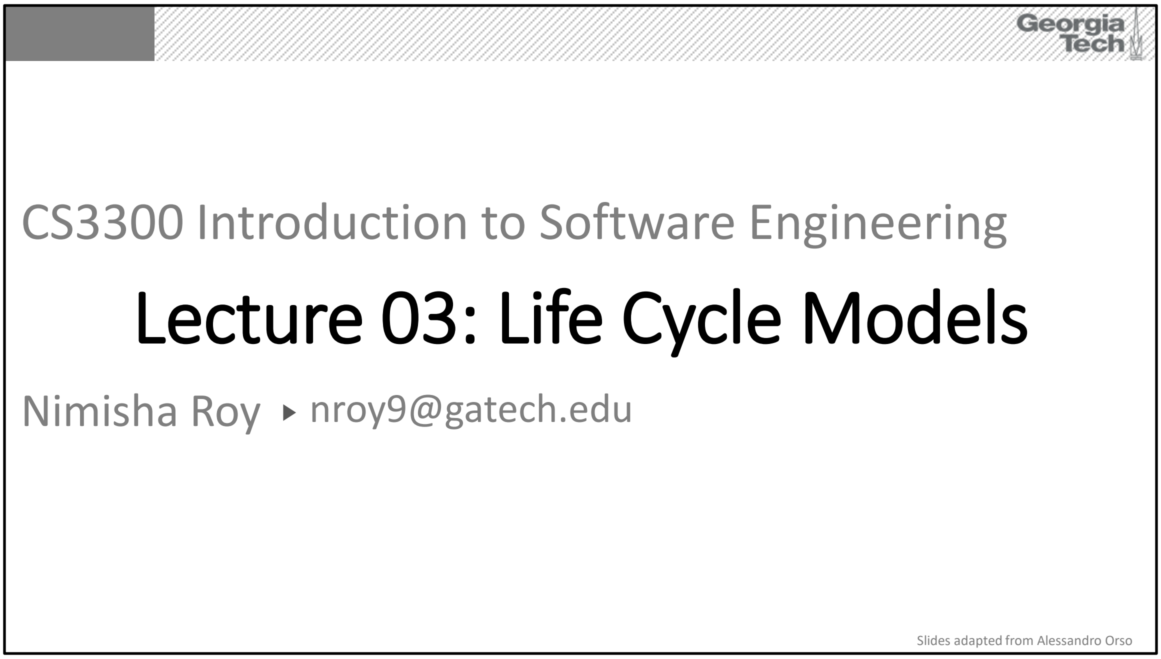


Announcements

- Group Mentors Assigned- You should have received an email
- Extra Credit Opportunity in today’s class
- Quiz 2 will release on 9/15 from 7 AM – 11:59 PM.
  - 10 minutes; Can take once; Using Honorlock;
  - Open physical notes; or browser tab restricted to Canvas, class website, lecture notes
  - Pool of questions; 15 random questions per student;
  - Based on class notes - GIT, life cycle model, RE, AJAX, REST, Maven



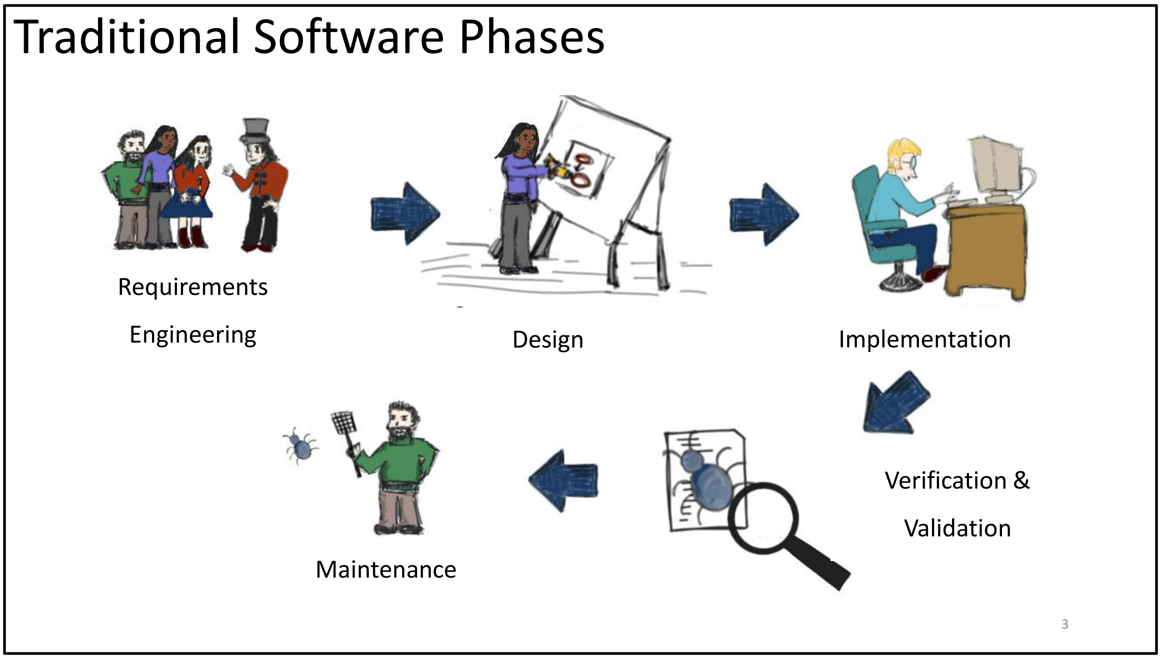
CS3300 Introduction to Software Engineering

Lecture 03: Life Cycle Models

Nimisha Roy ▸ nroy9@gatech.edu

Slides adapted from Alessandro Orso

Several traditional SE life cycle models, advantages, shortcomings, classic mistakes, Well-known ineffective development practices that when followed, tend to lead to bad results



Traditional Software Phases

Requirements Engineering

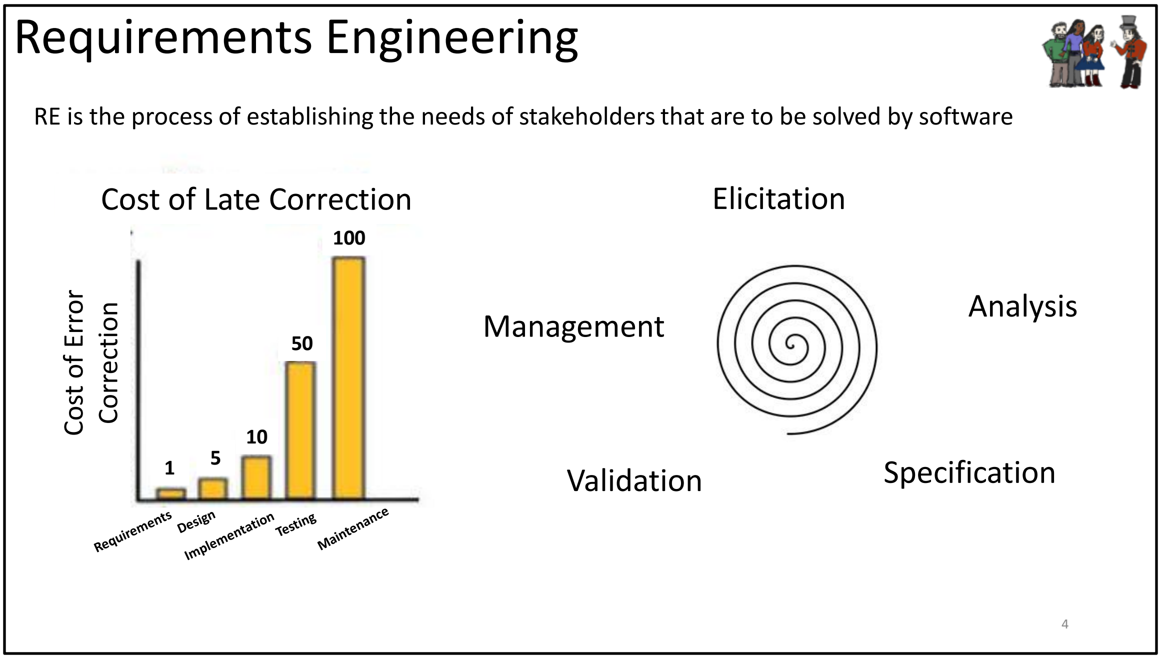
Design

Implementation

Verification & Validation

Maintenance

Generic activities in all software processes are:
Requirements and Specifications – talk to stakeholders, what the system should do
Design – define the design
Development - production of the SW system-write code
Verification and Validation - checking that the software does what it should do
Maintenance - changing the software in response to changing demands.
We will look at each approach in detail, how its done



Requirements Engineering

RE is the process of establishing the needs of stakeholders that are to be solved by software

Elicitation

Analysis

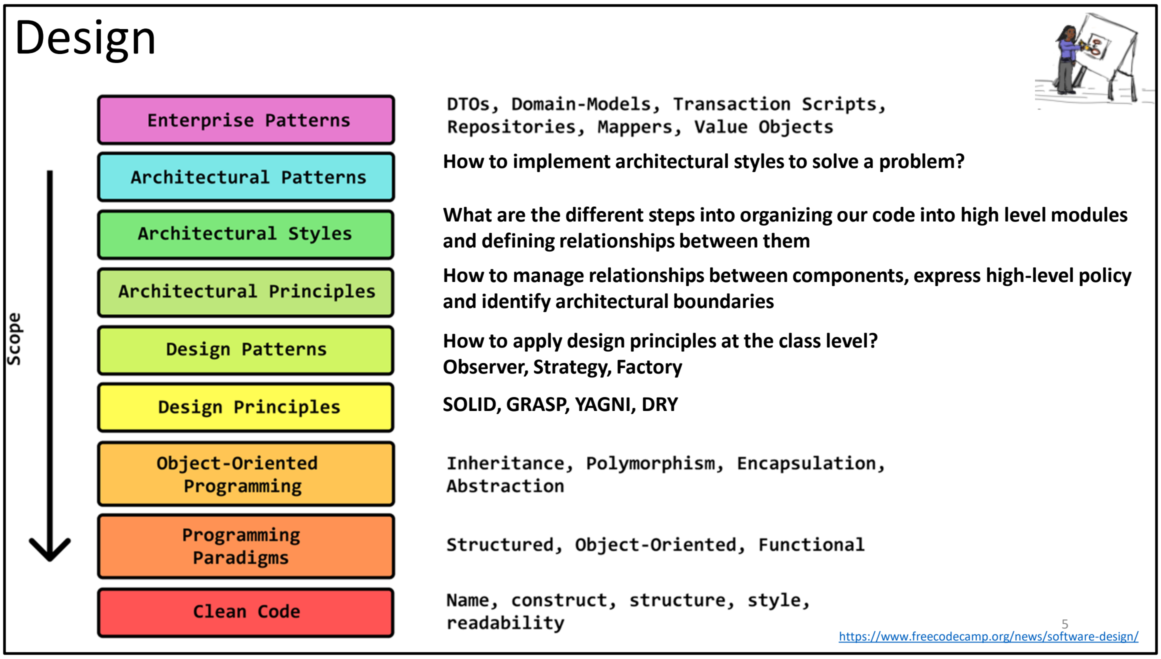
Specification

Validation

Management

4

Why important? Cost of correcting an error depends on number of subsequent decisions based on it. Error in understanding requirement potential for greatest cost since every other phase is dependent on it.
Cost grows dramatically. Collect requirements:
Elicitation: collection from stakeholders and other sources; analysis: study and deeper understanding of requirements; Specification: collective requirements are suitably represented, organized and saved so can be shared; Validation- complete, consistent, not redundant, satisfied important properties; management: account for changes during the lifetime. Not linear but iterative process.



Design

Scope

Enterprise Patterns	DTOs, Domain-Models, Transaction Scripts, Repositories, Mappers, Value Objects
Architectural Patterns	**How to implement architectural styles to solve a problem?**
Architectural Styles	**What are the different steps into organizing our code into high level modules and defining relationships between them**
Architectural Principles	**How to manage relationships between components, express high-level policy and identify architectural boundaries**
Design Patterns	**How to apply design principles at the class level?** **Observer, Strategy, Factory**
Design Principles	**SOLID, GRASP, YAGNI, DRY**
Object-Oriented Programming	Inheritance, Polymorphism, Encapsulation, Abstraction
Programming Paradigms	Structured, Object-Oriented, Functional
Clean Code	Name, construct, structure, style, readability

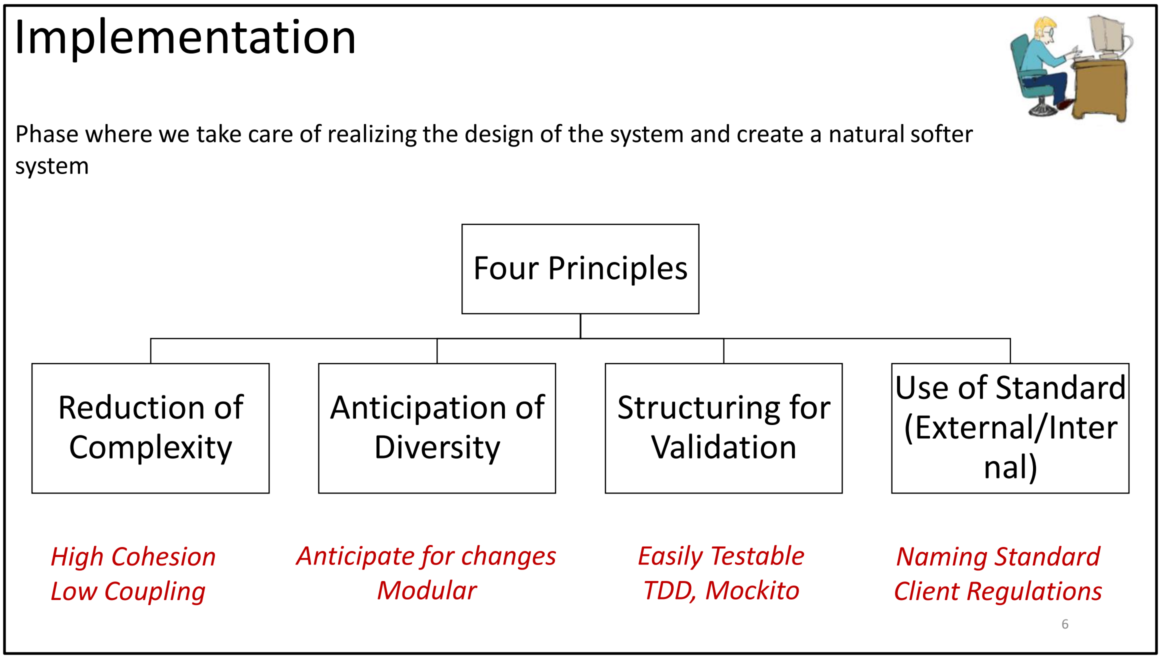
https://www.freecodecamp.org/news/software-design/

Design Patterns: Solutions to commonly solved design problems. How to apply the design principl

Enterprise patterns, often known as enterprise architecture patterns or enterprise design patterns, are best-practice solutions to recurring problems encountered during software development in an enterprise environment. They provide a standardized and reusable approach to common challenges, ensuring that solutions are both robust and scalable.

The concept is analogous to the well-known design patterns in software engineering (like the Singleton, Factory, and Observer patterns), but enterprise patterns typically address higher-level, more strategic problems related to system and application architectures, especially in large and complex enterprise settings.

es at the class level?



Implementation

Phase where we take care of realizing the design of the system and create a natural softer system

Four Principles

- Reduction of Complexity
- Anticipation of Diversity
- Structuring for Validation
- Use of Standard (External/Internal)

High Cohesion
Low Coupling

Anticipate for changes
Modular

Easily Testable
TDD, Mockito

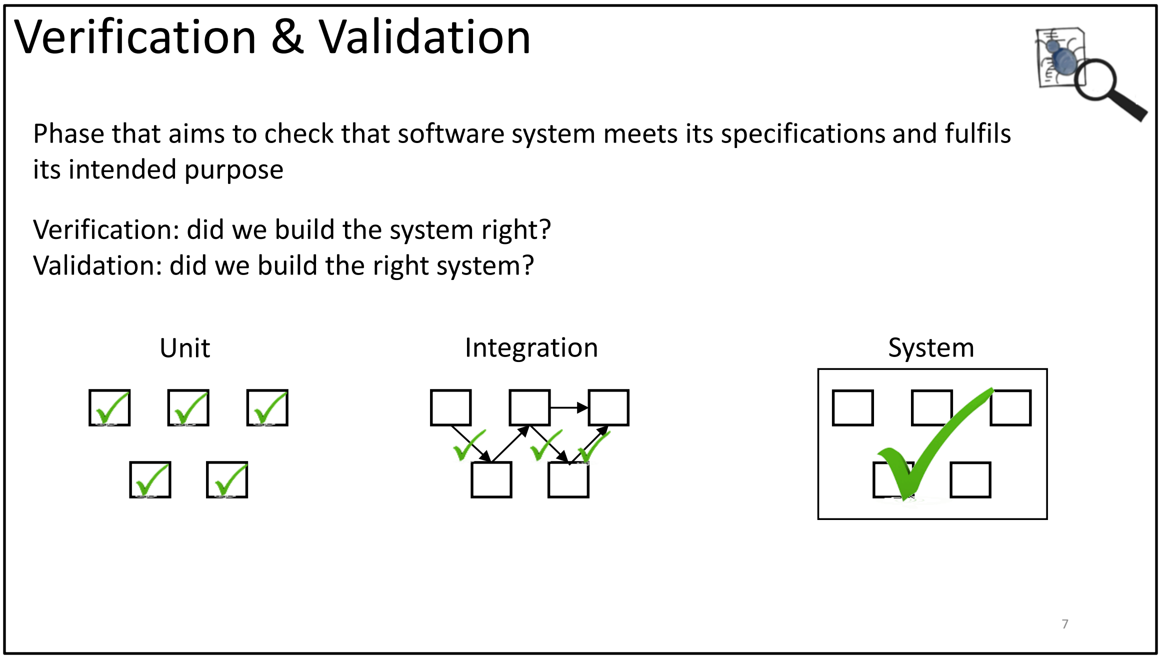
Naming Standard
Client Regulations

Reduction of complexity - aims to build software that is easier to understand and use.high cohesion, low coupling, comments

Anticipation of diversity - considers that software construction might evolve over time, anticipate and prepare for such changes. modular

Structuring for validation (aka design for testability) - build software so that it is easily testable during the subsequent validation and verification activities.

Use of standards - important that the software conforms to a set of internal or external standards and this is especially true within specific organizations and or domains. Internal: coding/naming standards within an organization; External: medical software regulations



Verification & Validation

Phase that aims to check that software system meets its specifications and fulfils its intended purpose

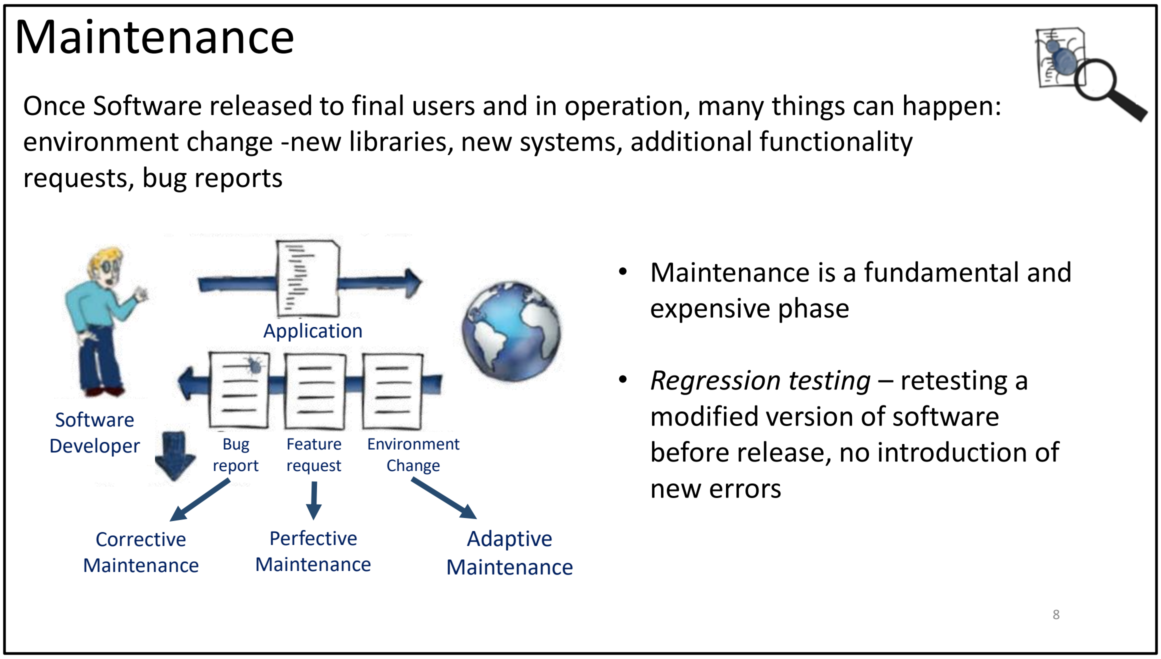
Verification: did we build the system right?
Validation: did we build the right system?

Unit

Integration

System

Test that individual units work as expected, interaction between units or the integration level is as expected, the system as a whole works as expected, all the different pieces work together the right way; this is also the place where we can do validation testing (stress/robustness testing)



Maintenance

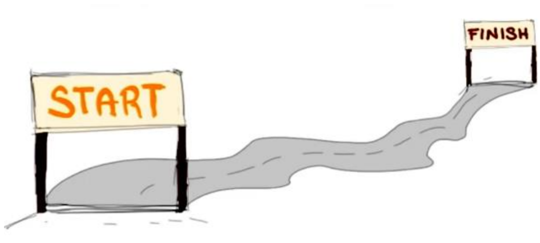
Once Software released to final users and in operation, many things can happen: environment change -new libraries, new systems, additional functionality requests, bug reports

- Maintenance is a fundamental and expensive phase
- *Regression testing* – retesting a modified version of software before release, no introduction of new errors

8

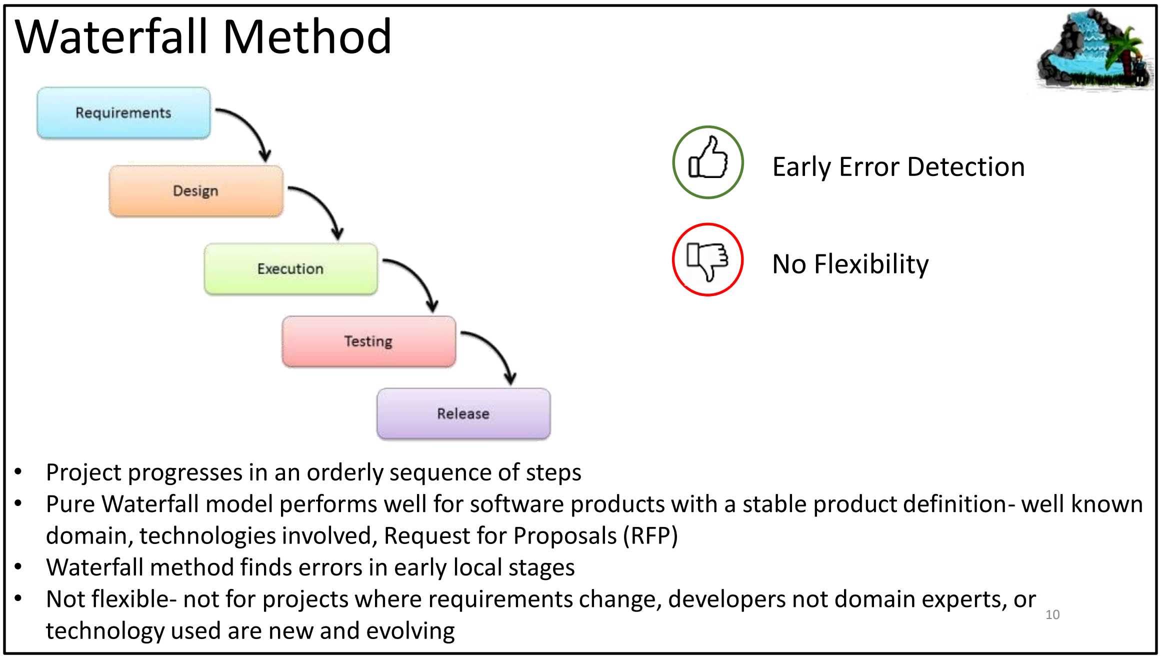
Sustains the product as it evolves through its life cycle. Corrective maintenance to eliminate problems with the code; Perfective maintenance to accommodate feature request, improve the software, make it more efficient; Adaptive maintenance, to take care of the environment changes. New version release.

Software Process Model/ Life Cycle Model



Functions:

- Order of activities
- Transition Criteria between Activities
- What should we do next and for how long?



Waterfall Method

Requirements → Design → Execution → Testing → Release

👍 Early Error Detection

👎 No Flexibility

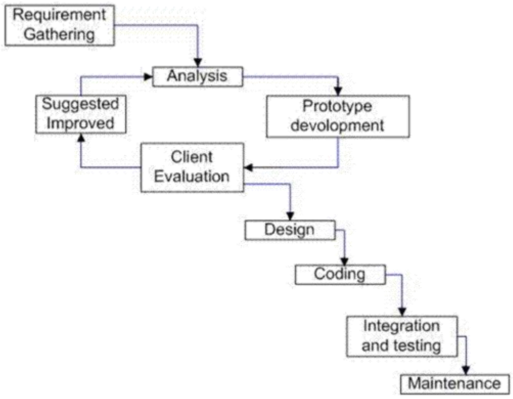
- Project progresses in an orderly sequence of steps
- Pure Waterfall model performs well for software products with a stable product definition- well known domain, technologies involved, Request for Proposals (RFP)
- Waterfall method finds errors in early local stages
- Not flexible- not for projects where requirements change, developers not domain experts, or technology used are new and evolving

Grandfather of all life cycle models. Review at the end of each phase. Difficult to fully specify requirements at the beginning of a project

•**A Human Life Is At Stake And A System Failure Could Result In Fatalities**

•**Money And Time Are Secondary Factors And What Matters More Is The Safety And Stability Of A Project**

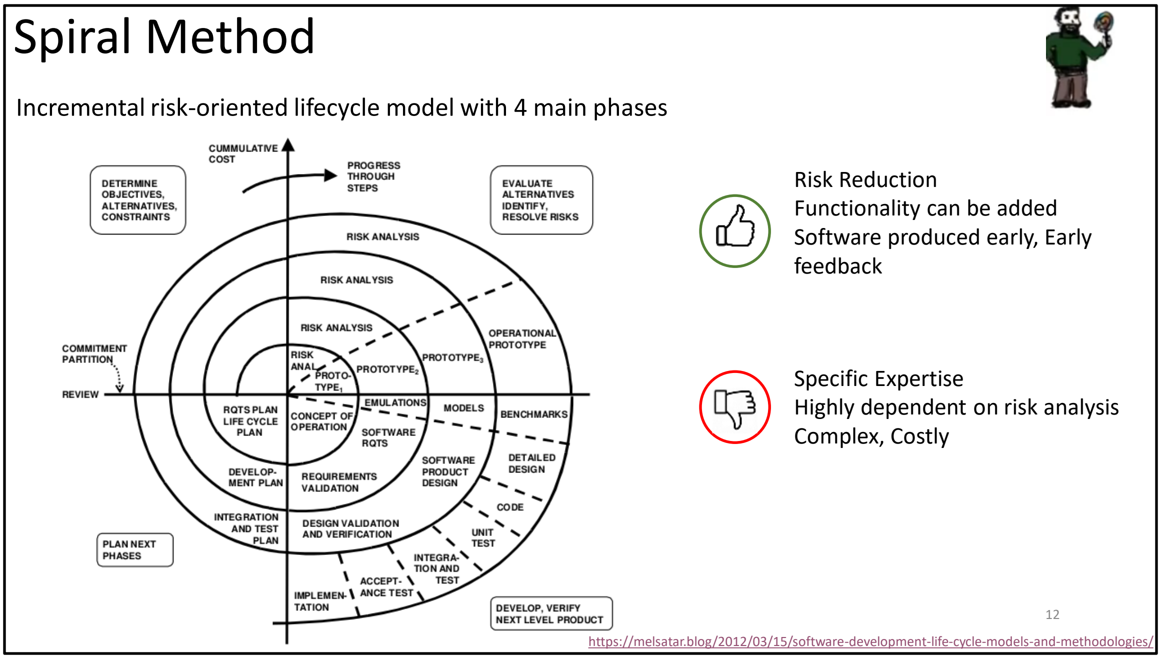
Evolutionary Prototyping



Immediate feedback
Helps Requirements understanding

Difficult to Plan
Can deteriorate to code-and-fix

- Prototypes that evolve into the final system through an iterative incorporation of user feedback.
- Ideal when not all requirements are well-understood. System keeps evolving based on customer feedback



Spiral Method

Incremental risk-oriented lifecycle model with 4 main phases

👍 Risk Reduction
Functionality can be added
Software produced early, Early feedback

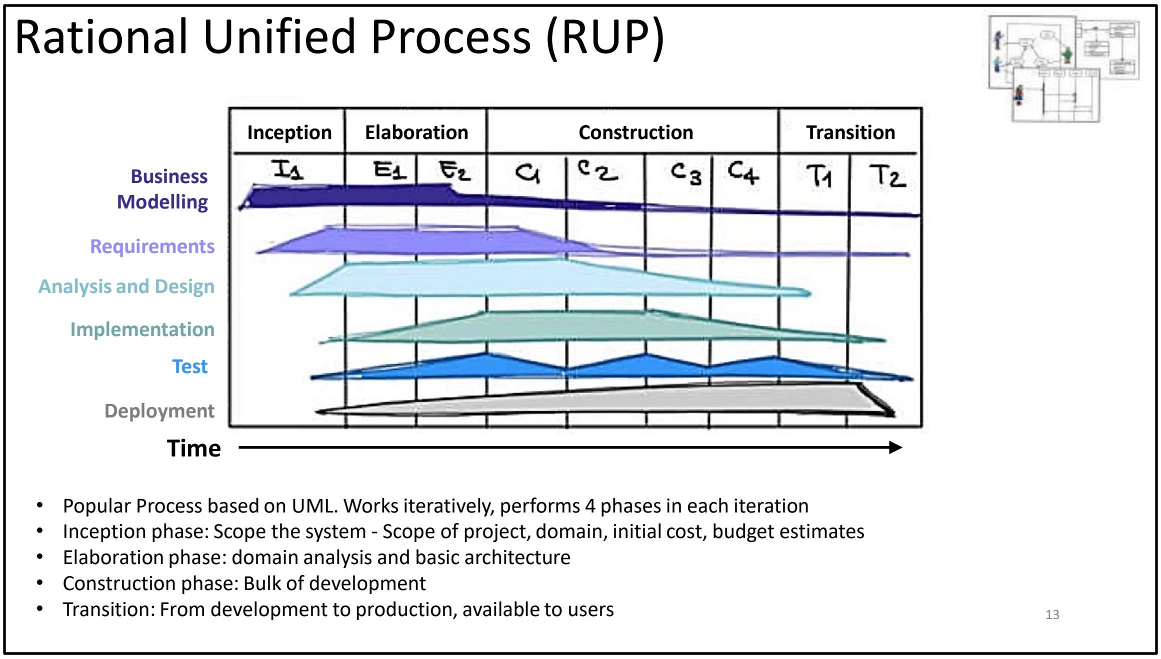
👎 Specific Expertise
Highly dependent on risk analysis
Complex, Costly

12

https://melsatar.blog/2012/03/15/software-development-life-cycle-models-and-methodologies/

- **Determine objectives** - the requirements will be gathered
- **Identify and resolve risks** - the risks and the alternate solutions will be identified, and a prototype will be produced
- **Development and tests** – produce software and tests for the software
- **Plan the next iteration** - the output of the project, so far, is evaluated, and the next iteration is planned

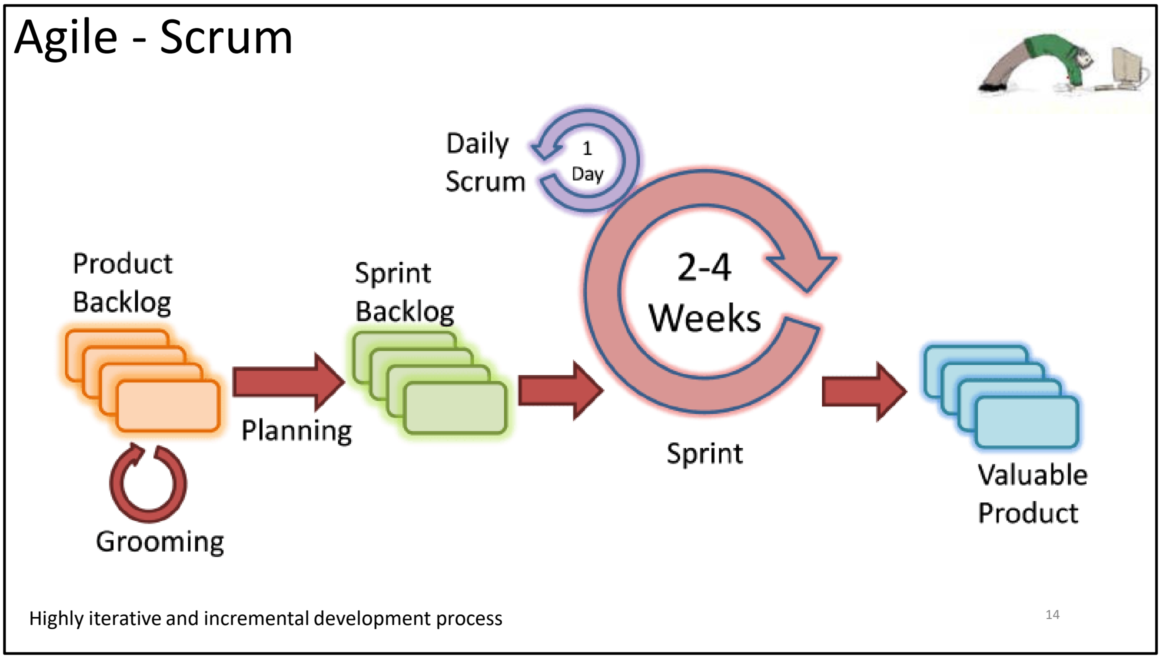
- Risk analysis needs expertise



Rational Unified Process (RUP)

- Popular Process based on UML. Works iteratively, performs 4 phases in each iteration
- Inception phase: Scope the system - Scope of project, domain, initial cost, budget estimates
- Elaboration phase: domain analysis and basic architecture
- Construction phase: Bulk of development
- Transition: From development to production, available to users

In between fixed and iterative. Can be customized by the development organization according to their needs. Slightly similar to waterfall, it has fixed phases as inception, elaboration, construction and transition. But unlike waterfall, RUP is an iterative process.



Agile - Scrum

Daily Scrum

1 Day

Product Backlog

Grooming

Planning

Sprint Backlog

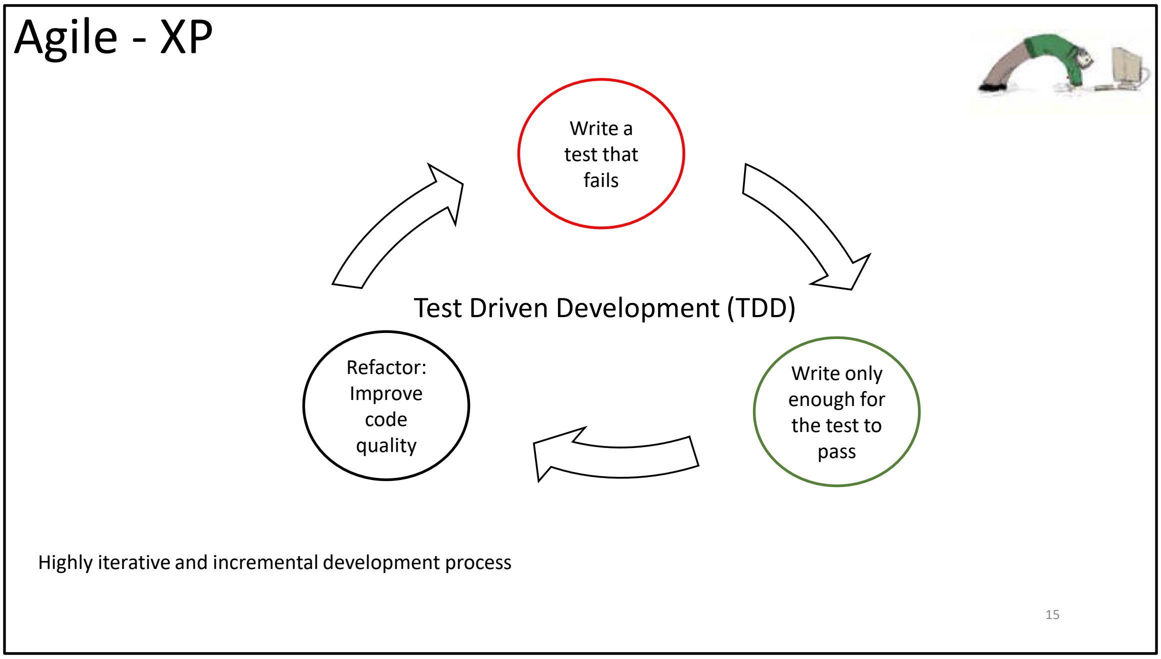
2-4 Weeks

Sprint

Valuable Product

Highly iterative and incremental development process

Readable, maintainable



Agile - XP

Test Driven Development (TDD)

Write a test that fails

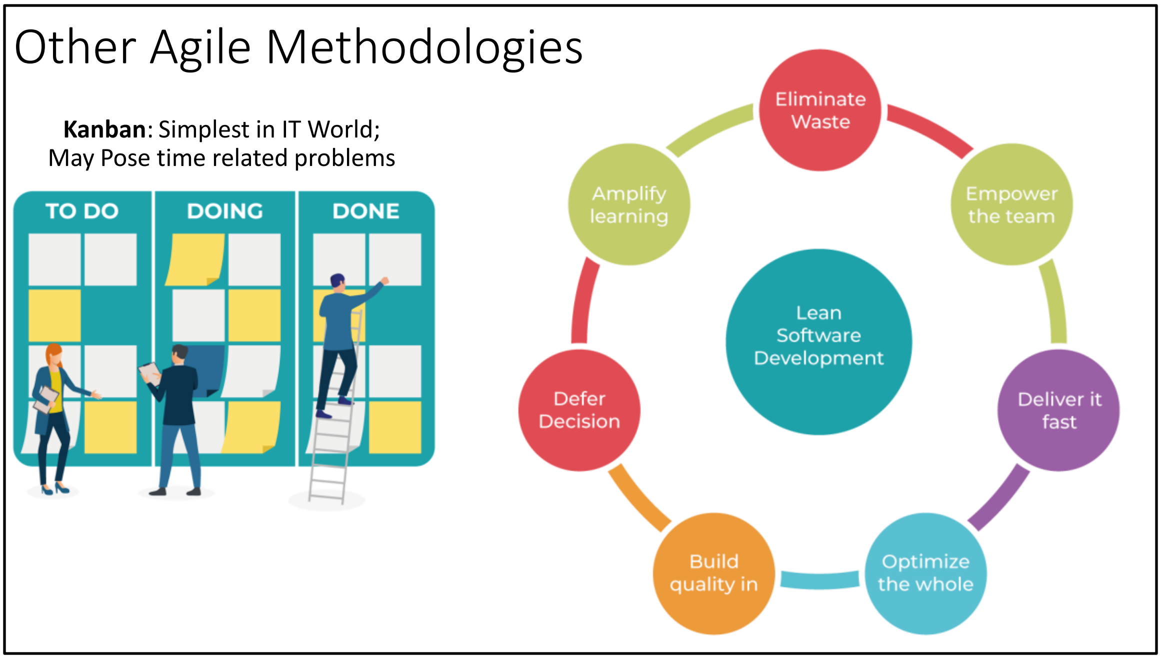
Write only enough for the test to pass

Refactor: Improve code quality

Highly iterative and incremental development process

15

Readable, maintainable



Other Agile Methodologies

Kanban: Simplest in IT World;
May Pose time related problems

Lean Software Development

Eliminate Waste

Empower the team

Deliver it fast

Optimize the whole

Build quality in

Defer Decision

Amplify learning

Great for remote-first teams; visual No timeframes

ruthlessly remove any activity that does not bring ultimate value to the product, tams struggling with efficiency issues

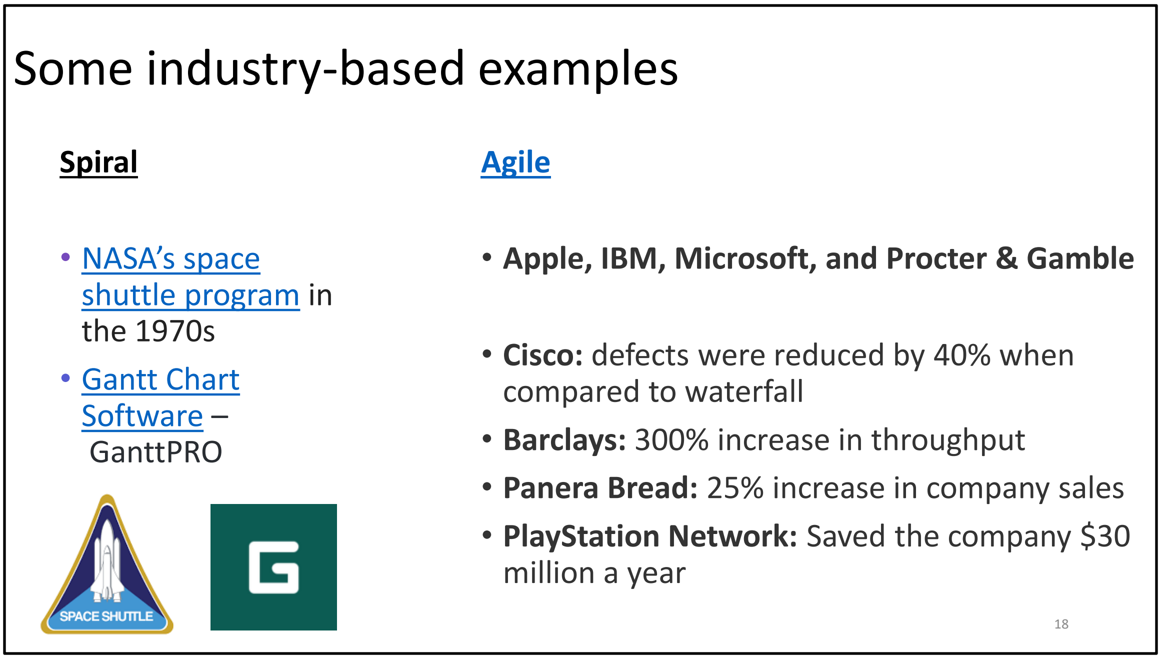
Some industry-based examples

Waterfall

Military And Aircraft Programs Where Requirements Are Declared Early On And Remain Constant

Evolutionary Prototyping

- **Company**: **Broderbund Software**.
- **Project**: The creation of the original **"Prince of Persia"** video game. The initial version of the game was created and then improved upon based on feedback and playtesting.



Some industry-based examples

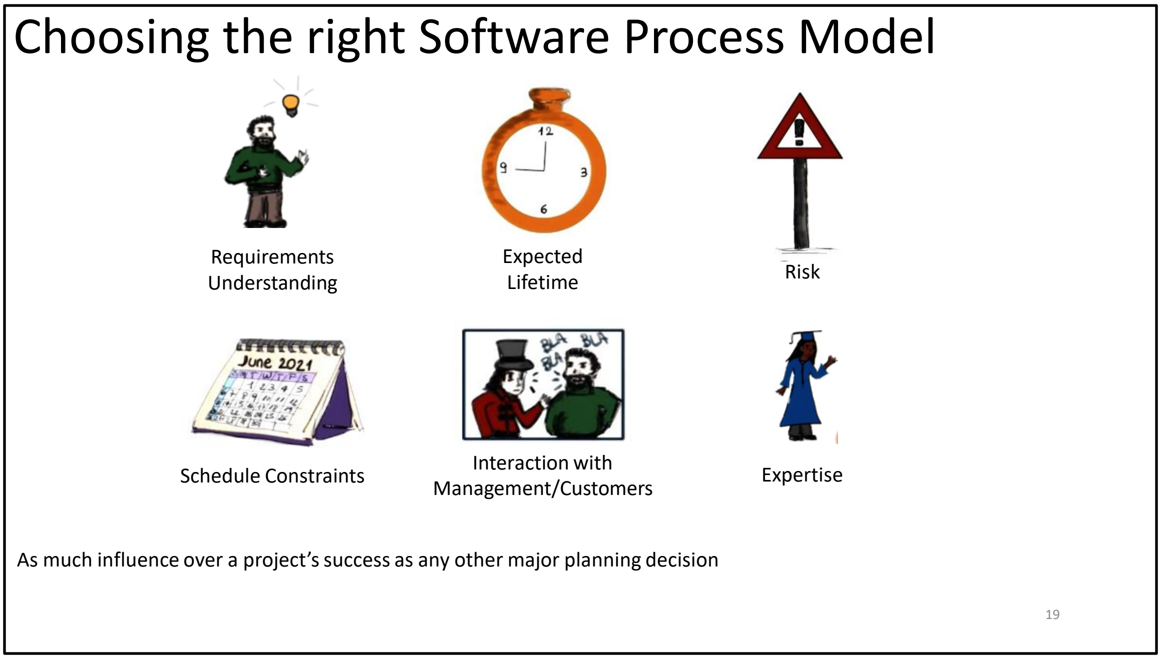
Spiral

- NASA's space shuttle program in the 1970s
- Gantt Chart Software – GanttPRO

SPACE SHUTTLE

Agile

- **Apple, IBM, Microsoft, and Procter & Gamble**
- **Cisco:** defects were reduced by 40% when compared to waterfall
- **Barclays:** 300% increase in throughput
- **Panera Bread:** 25% increase in company sales
- **PlayStation Network:** Saved the company $30 million a year



Choosing the right Software Process Model

Requirements Understanding

Expected Lifetime

Risk

Schedule Constraints

Interaction with Management/Customers

Expertise

As much influence over a project's success as any other major planning decision

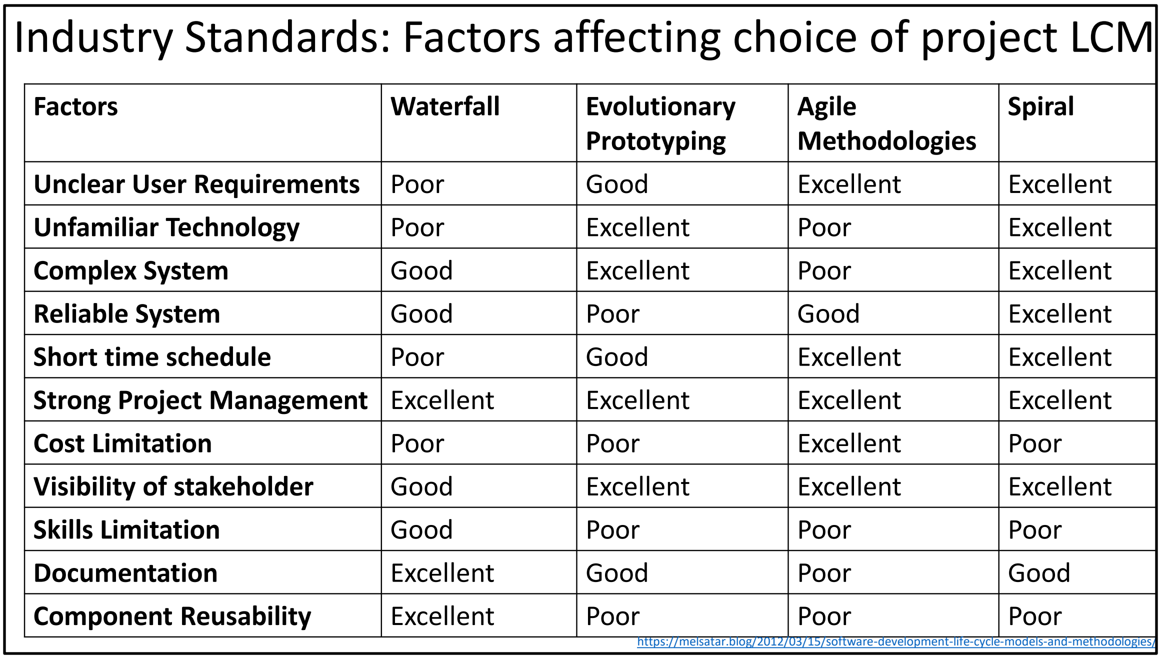
19

Do we understand all requirements in advance? If yes, waterfall could be a choice, else EP
Quick project for specific purpose or something that will last longer and will need maintenance.
Do we know the domain well, technologies involved? If so, waterfall, if not, agile could be a choice
How much time and resources? Lot of time- Spiral
Processes like EP, spiral require constant input from customer
Do we have people who know all technologies needed, processes etc. Spiral needs risk analysis expertise.

Industry Standards: Factors affecting choice of project LCM

- Degree of Project Complexity
- Work/Time Flexibility
- Project Focus/ Client involvement
- Size of organization
- Role Specialization
- Budget

https://asana.com/resources/project-management-methodologies
https://thedigitalprojectmanager.com/projects/pm-methodology/project-management-methodologies-made-simple/



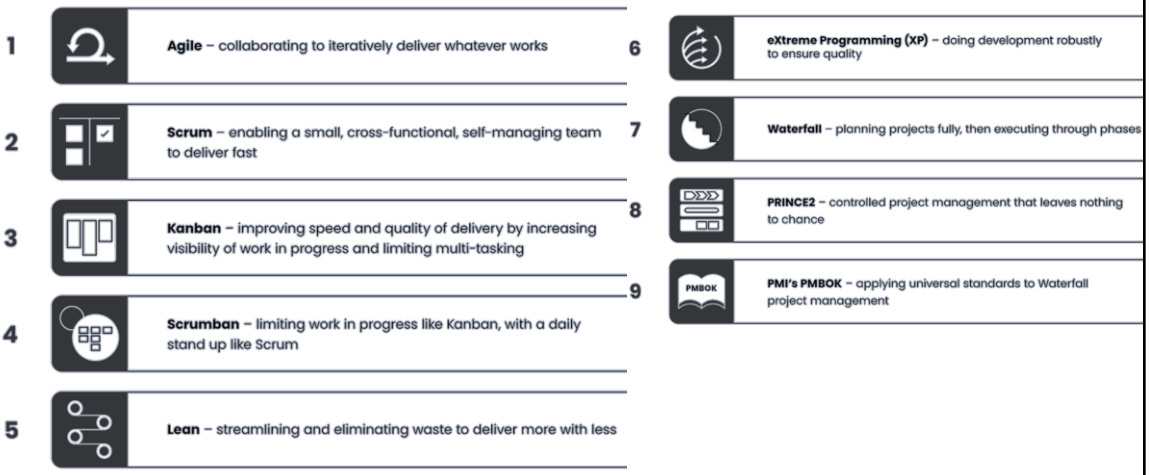
Industry Standards: Factors affecting choice of project LCM

Factors	**Waterfall**	**Evolutionary Prototyping**	**Agile Methodologies**	**Spiral**
Unclear User Requirements	Poor	Good	Excellent	Excellent
Unfamiliar Technology	Poor	Excellent	Poor	Excellent
Complex System	Good	Excellent	Poor	Excellent
Reliable System	Good	Poor	Good	Excellent
Short time schedule	Poor	Good	Excellent	Excellent
Strong Project Management	Excellent	Excellent	Excellent	Excellent
Cost Limitation	Poor	Poor	Excellent	Poor
Visibility of stakeholder	Good	Excellent	Excellent	Excellent
Skills Limitation	Good	Poor	Poor	Poor
Documentation	Excellent	Good	Poor	Good
Component Reusability	Excellent	Poor	Poor	Poor

https://melsatar.blog/2012/03/15/software-development-life-cycle-models-and-methodologies/

Project planning

Industry Standards: Most Popular Methods



1 **Agile** – collaborating to iteratively deliver whatever works

2 **Scrum** – enabling a small, cross-functional, self-managing team to deliver fast

3 **Kanban** – improving speed and quality of delivery by increasing visibility of work in progress and limiting multi-tasking

4 **Scrumban** – limiting work in progress like Kanban, with a daily stand up like Scrum

5 **Lean** – streamlining and eliminating waste to deliver more with less

6 **eXtreme Programming (XP)** – doing development robustly to ensure quality

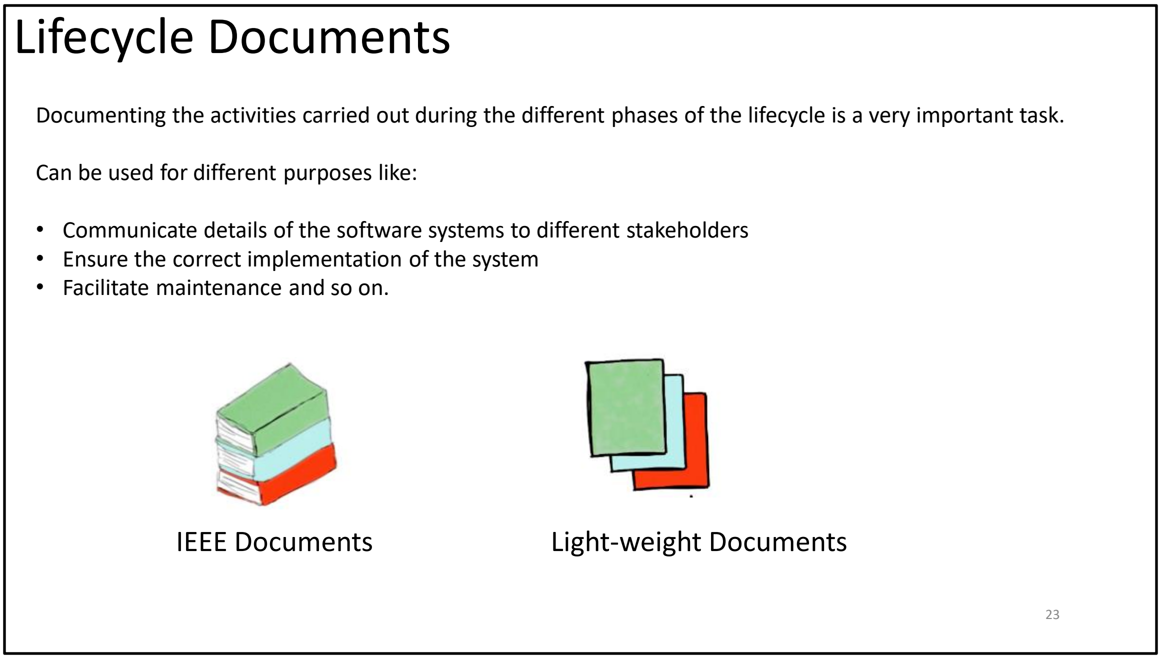
7 **Waterfall** – planning projects fully, then executing through phases

8 **PRINCE2** – controlled project management that leaves nothing to chance

9 **PMI's PMBOK** – applying universal standards to Waterfall project management

https://thedigitalprojectmanager.com/projects/pm-methodology/project-management-methodologies-made-simple/

Project Management Body of Knowledge



Lifecycle Documents

Documenting the activities carried out during the different phases of the lifecycle is a very important task.

Can be used for different purposes like:

- Communicate details of the software systems to different stakeholders
- Ensure the correct implementation of the system
- Facilitate maintenance and so on.

IEEE Documents
Light-weight Documents

Standardized documents by IEEE, heavyweight, HWs have lightweight implementation

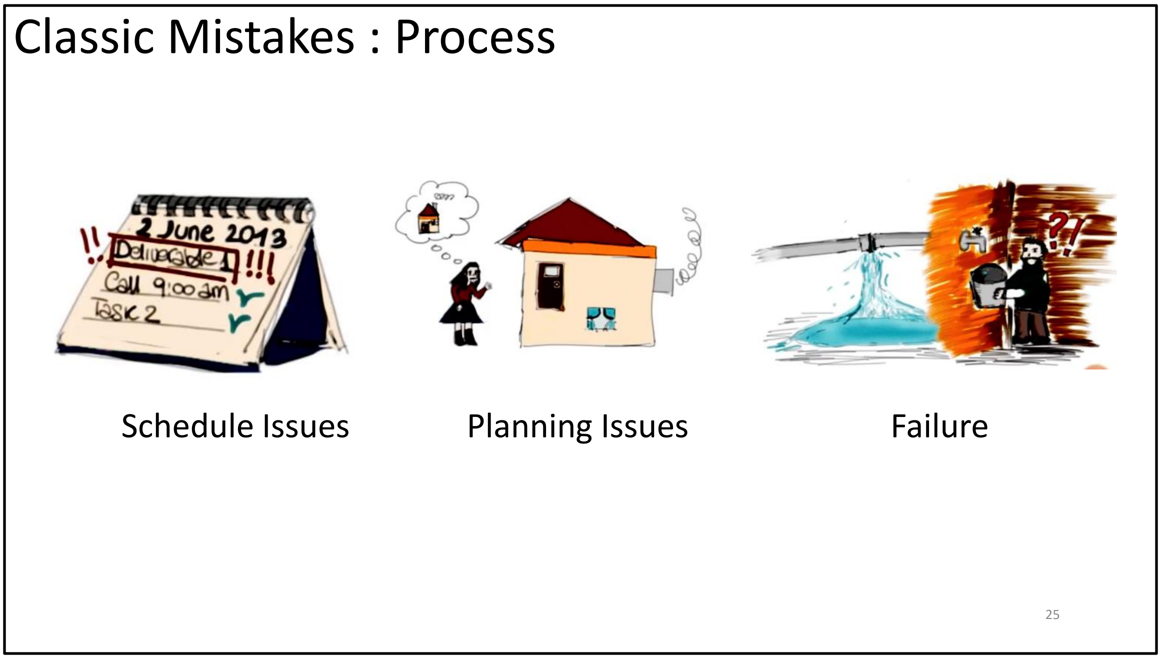
Classic Mistakes : People

Heroics

Work Environment

People Management

Ineffective practices
Can do attitude, 1 person can do everything , discourages cooperation. Work in teams
Productivity increases when workplace is nice, quiet, warm and welcoming
Lack of leadership, using wrong means in wrong ways, adding people when deadline not met



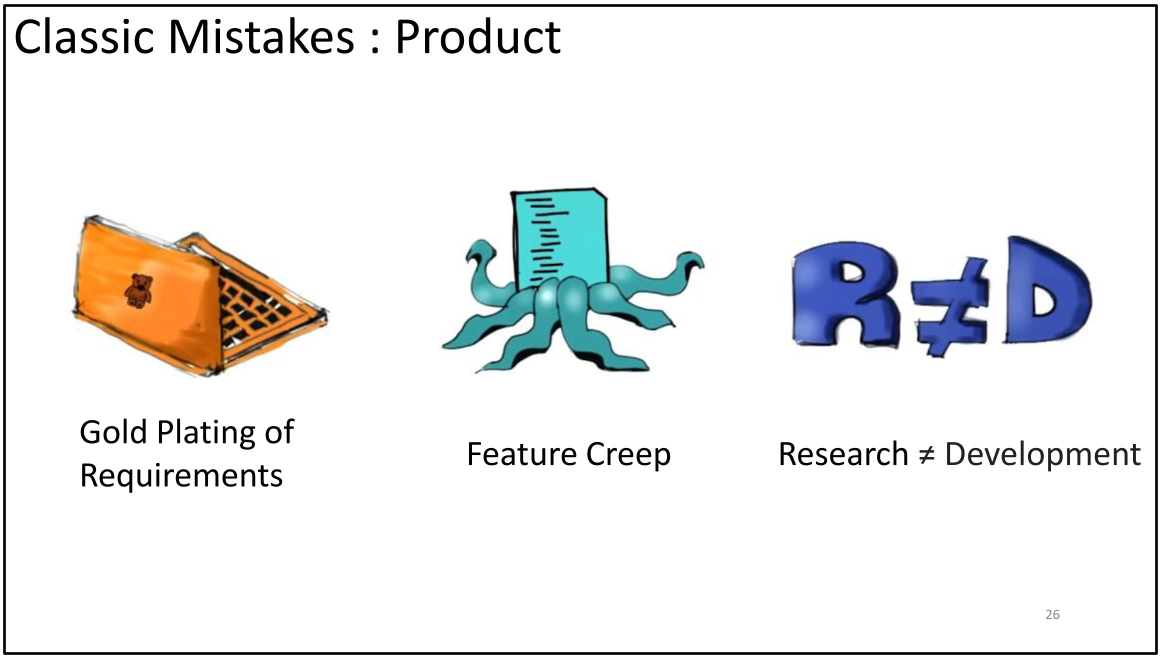
Classic Mistakes : Process

Schedule Issues

Planning Issues

Failure

Can't come up with a realistic schedule, overly optimistic schedule because we underestimate the effort involved in different parts of project, overestimate the ability of people, overestimate the importance of use of tools, result is late project
Insufficient planning
Unforeseen failures at one place affects downstream activities



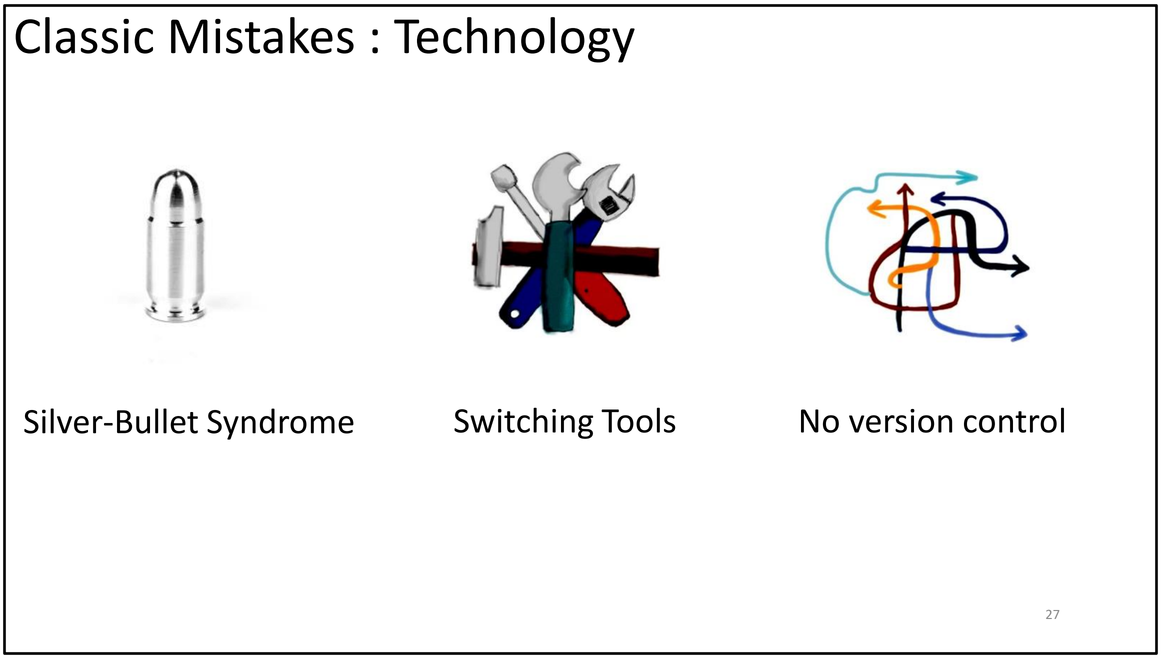
Classic Mistakes : Product

Gold Plating of Requirements

Feature Creep

Research ≠ Development

More features than needed, by marketing, lengthens schedule
Adding more features , avg project experiences 25% growth in features over its lifetime
Might need more research



Classic Mistakes : Technology

Silver-Bullet Syndrome

Switching Tools

No version control

Should not rely too much on technology. there is too much reliance on the advertised benefits of some previously unused technology.

Introducing new tools would need a steep learning curve, upgrading tools make sense

Quizizz