IS2001: Software Engineering

2. Software Development Process Models -2

Viraj Welgama

Course Outline

1. Introduction
2. Software Process Models (2)
3. Requirement Analysis and Specification (2)
4. Software Design
5. Coding
6. Software Testing
7. Maintenance
8. Software Project Management
9. Software Quality Management

Learning Objectives

- Describe different process models used for software development
- Identify the most appropriate software process model for a given problem
- Identify how CASE tools can be used to support software process activities

Recap…

- Why we need a process?
- Generic activities
  - Specification, Design & Development, Validation, Evolution
- Process Models
  - Waterfall, Prototyping, Incremental Development (RAD), Spiral Model, Unified Process, Agile Software Development (XP, Scrum)

Recap…

- Why we need a process?
- Generic activities
  - Specification, Design & Development, Validation, Evolution
- Process Models
  - Waterfall, Prototyping, Incremental Development (RAD), Spiral Model, Unified Process, Agile Software Development (XP, Scrum)

Unified Process (UP)

"Use-case Driven, Architecture-Centric, Iterative and Incremental" Software Development Process Model

- Use-case Driven?
  - Recognize Customer View and Customer Communication
- Architecture Centric?
  - Emphasize important role of software architecture
  - Understandability, Reliance to future changes, Reuse
- Process: Iterative and Incremental

Incremental?

- *Incremental development* is a scheduling and staging strategy in which the various parts of the system are developed at different times or rates, and integrated as they are completed.
- The alternative to incremental development is to develop the entire system with a "big bang" integration.

Iterative?

- *Iterative development* is a rework scheduling strategy in which time is set aside to revise and improve parts of the system.
- A typical difference is that the output from an increment is released to users, whereas the output from an iteration is examined for modification.
- For example, an incremental release might include new functions, by contrast, an iteration is more likely to be integrated, examined, tested, with the result be marked as work in the following iteration, all prior to release.

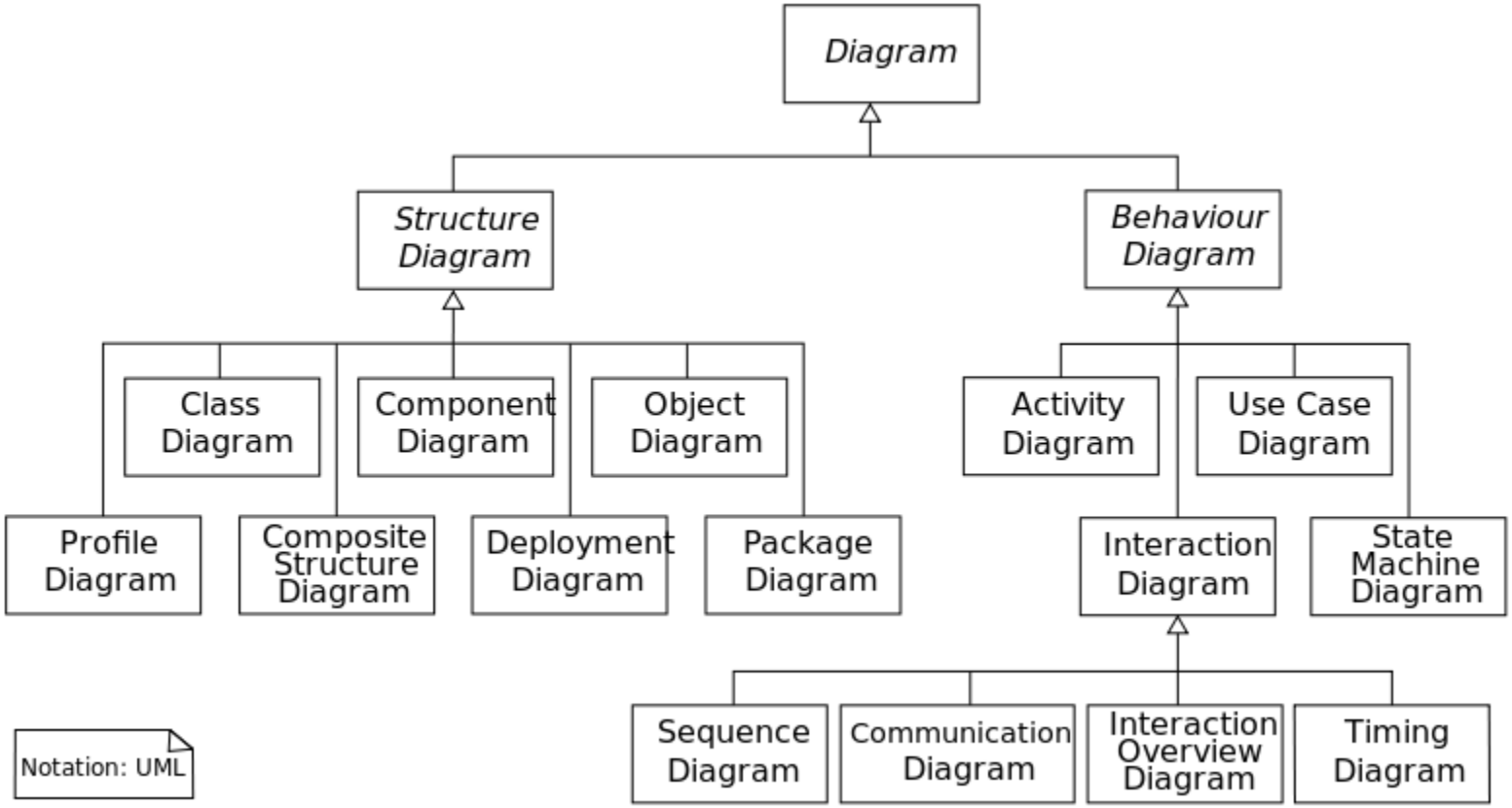
UP: History / Context

- Object-Oriented Approach (early 1990) for analysis and design
- UML common tool (Unified Modeling Language)
  - Provide Necessary technology to support OO Software Engineering practice
  - Not a process framework to guide the project team to apply OO practices
- UML and UP are used together

UML

- The Unified Modeling Language (UML) is a modeling language
- Designed to provide a standard way to visualize the design of a system.
  - Any activities (jobs)
  - Individual components of the system
  - And how they can interact with other software components.
  - How the system will run
  - How entities interact with others (components and interfaces)
  - External user interface

UML 2 Diagram types



Unified Process

- UP identifies the best features and characteristics of conventional software process models and integrate them with the best principles/practices of agile software development

UP: Use Case Driven

- In the Unified Process, use cases are used to capture the functional requirements and to define the contents of the iterations.
- Each iteration takes a set of use cases or scenarios from requirements all the way through implementation, test and deployment.

UP: Architecture Centric

- The Unified Process insists that architecture sit at the centre of the project.
- Since no single model is sufficient to cover all aspects of a system, the Unified Process supports multiple architectural models and views.
- One of the most important deliverables of the process is the executable architecture baseline which is created during the Elaboration phase. This partial implementation of the system serves to validate the architecture and act as a foundation for remaining development.

UP: Risk focused

- The Unified Process requires the project team to focus on addressing the most *critical risks early in the project life cycle*.

- In the Elaboration phase, deliverables must be selected in order to ensure that *the biggest risks are addressed first.*

Phases in UP

- Four Phases
  - Inception
  - Elaboration
  - Construction
  - Transition
- Based on Generic Framework phases/activities

Inception Phase

- Encompasses customer communication and planning activities
  - Identify business requirements
    - Initial Use Cases (functions, features, actors, sequence of actions, basis for project plan)
  - Initial (Rough) Architecture of the System
    - Outline of Major sub-systems, functions, features
    - Refine to model different views of the system
  - Plan for number of incremental and iterations
    - Identify resources, risks, schedule, basis for phases

Inception Phase: Goals

- Establish a justification or business case for the project
- Establish the project scope and boundary conditions
- Outline the use cases and key requirements that will drive the design tradeoffs
- Outline one or more candidate architectures
- Identify risks
- Prepare a preliminary project schedule and cost estimate

Elaboration Phase

- Objectives
  - Capture as many as system requirements
  - Address known risks
  - Establish and Validate System Architecture
- Customer Communication and Major Modeling Activities
- Refine initial use-cases to establish 5 models
  - Use-case Model
  - Analysis Model
  - Design Model
  - Implementation Model
  - Deployment Model

Elaboration Phase

- Create “Executable Architectural Baseline”
  - Justify viability of system irrespective of all functions and features. A partial implementation to validate the architecture and act as a foundation for remaining development.
- Review the plan
  - Scope
  - Risks
  - Delivery Dates

  If necessary, modify the plan as necessary

Construction Phase

- Construction is the largest phase in the project.
- the remainder of the system is built on the foundation laid in Elaboration.
- System features are implemented in a series of short, time-boxed iterations.
- Each iteration results in an executable release of the software.

Construction Phase

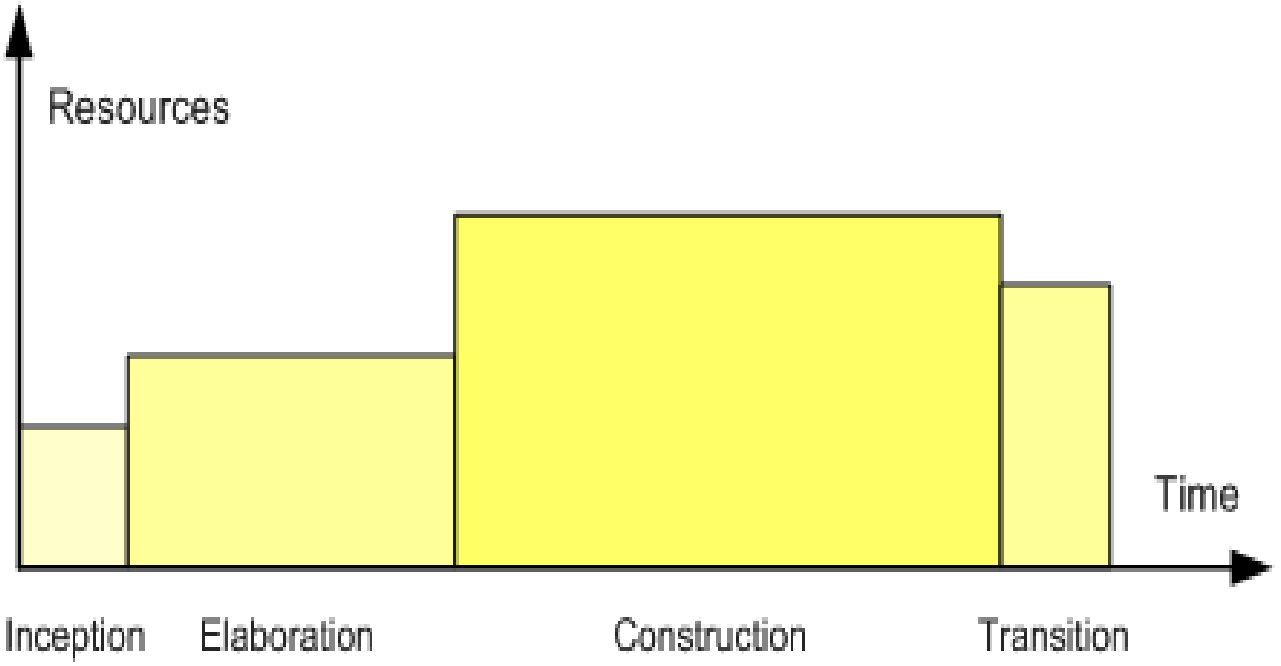
- Make each use case an operation in the system (Increment)
  - All increments started in elaboration phase to be completed
  - Develop test cases for Acceptance test
  - Unit Testing for each Increment as and when completed
  - Integration activities (component assembly and Integration Testing)

Transition Phase

- Beta Testing (Software is given to users)
- Gather users feedback (errors and changes required)
- Develop supporting materials
  - User manuals, trouble-shooting guidelines, installation procedures etc.
- At the end, the release should be a usable product
- Includes system conversions and user training.

UP: Resources

- Resources required in the cycle



Major work products produced for each UP phases

Inception Phase	Elaboration Phase
Vision document	Use-case model
Initial use-case model	Requirements functional, non-functional
Initial risk assessment	Analysis model
Project Plan	Software architecture
Business model	Preliminary design model
Prototypes	Revised risk list, revised prototypes
Construction Phase	**Transition Phase**
Design model	Delivered software increment
Software components	Beta test reports
Integrated software increment	General user feedback
Test cases	
Test Plan and Procedures	
Support documentation	

Rational Unified Process (RUP)

- **The Rational Unified Process (RUP)** is an iterative software development process framework created by the Rational Software Corporation, a division of IBM
- RUP is a specific implementation of the Unified Process.

Different variants of the Unified Process

- Agile Unified Process (AUP), a lightweight variation developed by Scott W. Ambler
- Basic Unified Process (BUP), a lightweight variation developed by IBM and a precursor to OpenUP
- Enterprise Unified Process (EUP), an extension of the Rational Unified Process
- Essential Unified Process (EssUP), a lightweight variation developed by Ivar Jacobson
- Open Unified Process (OpenUP), the Eclipse Process Framework software development process
- Rational Unified Process (RUP), the IBM / Rational Software development process
- Oracle Unified Method (OUM), the Oracle development and implementation process
- Rational Unified Process-System Engineering (RUP-SE), a version of RUP tailored by Rational Software for System Engineering

AGILE SOFTWARE DEVELOPMENT

Agile…

- Cambridge Dic;

  agile

  adjective /ˈædʒ·əl, -aɪl/ **US**

  › **able to move quickly and easily:**

- Princeton Wordnet;

  S: (adj) **agile**, nimble, quick, spry (moving quickly and lightly) *"sleek and agile as a gymnast"; "as nimble as a deer"; "nimble fingers"; "quick of foot"; "the old dog was so spry it was halfway up the stairs before we could stop it"*

Agile Software Development

- **Agile** is a group of software development methods which promote adaptive planning, evolutionary development, early delivery, continuous improvement and encourages rapid and flexible response to change.

- Requirements and solutions evolve through collaboration between self-organizing, cross-functional teams.

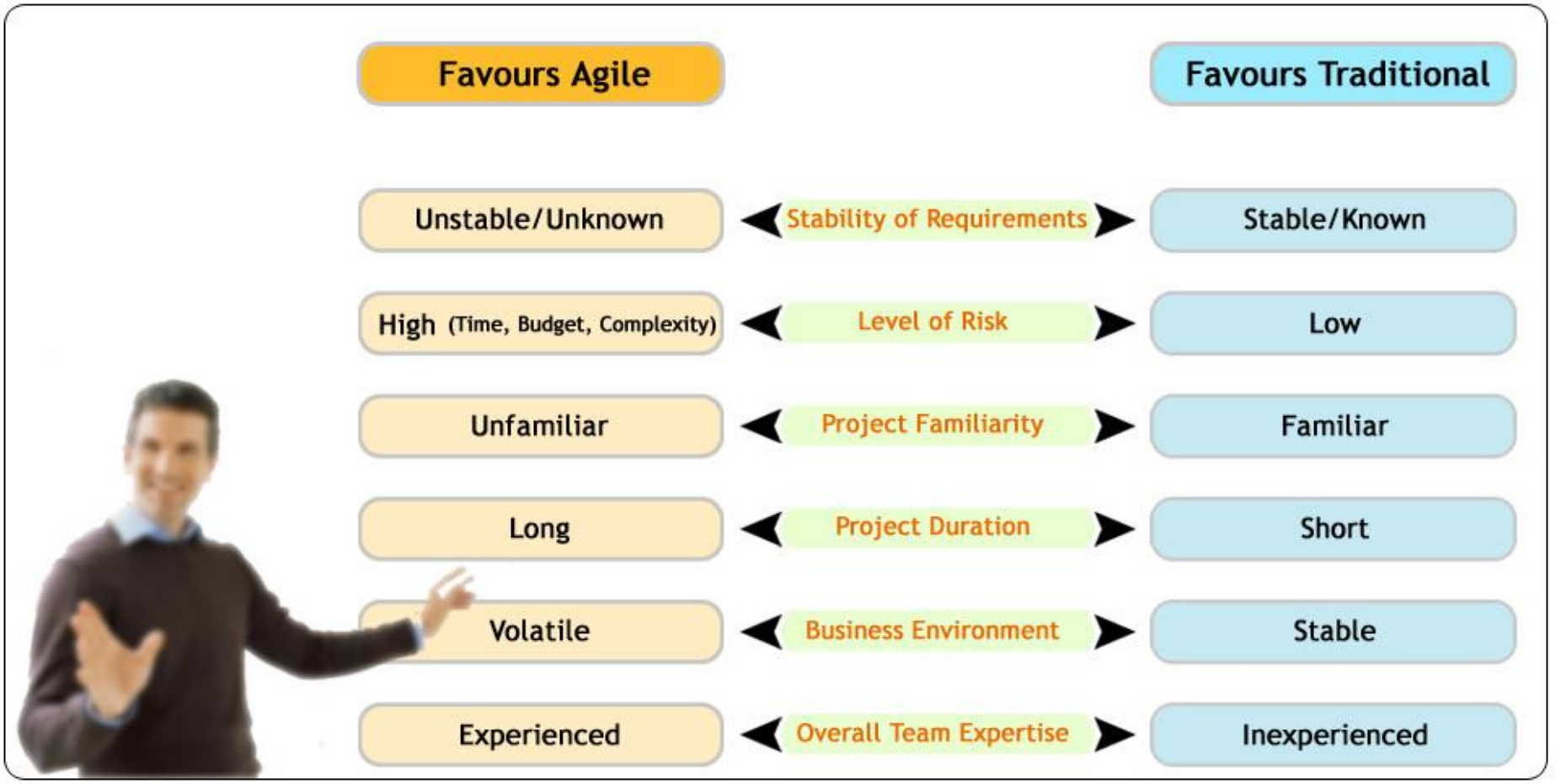
Agile Manifesto

- Individuals and interactions over processes and tools
- Working product over comprehensive documentation
- Customer collaboration over contact negotiation
- Responding to change over following a plan

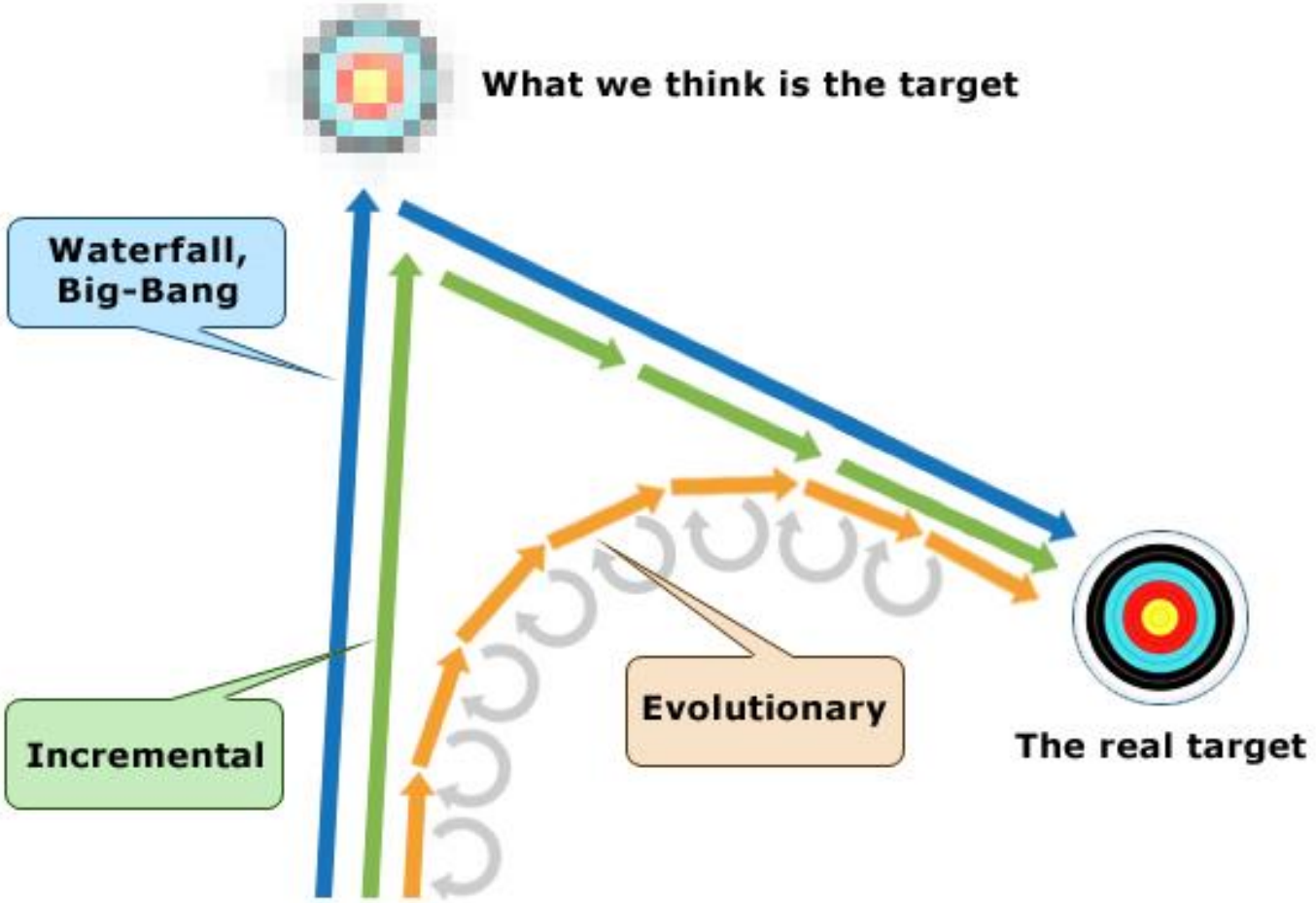
Features of Agile Processes

- Based on learn & adapt rather than rigid processes.
- Working software is the primary measure of progress. Working system releases in short iterations.
- Informal processes compared to traditional methods.
- Customers should be closely involved throughout the development process.
- Maintain simplicity in order to adapt to change.
- Small, skilled and empowered teams. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.
- Efficient (Face-to-face as much as possible) communication.
- Less documentation compared to traditional methods.

When to use Agile?

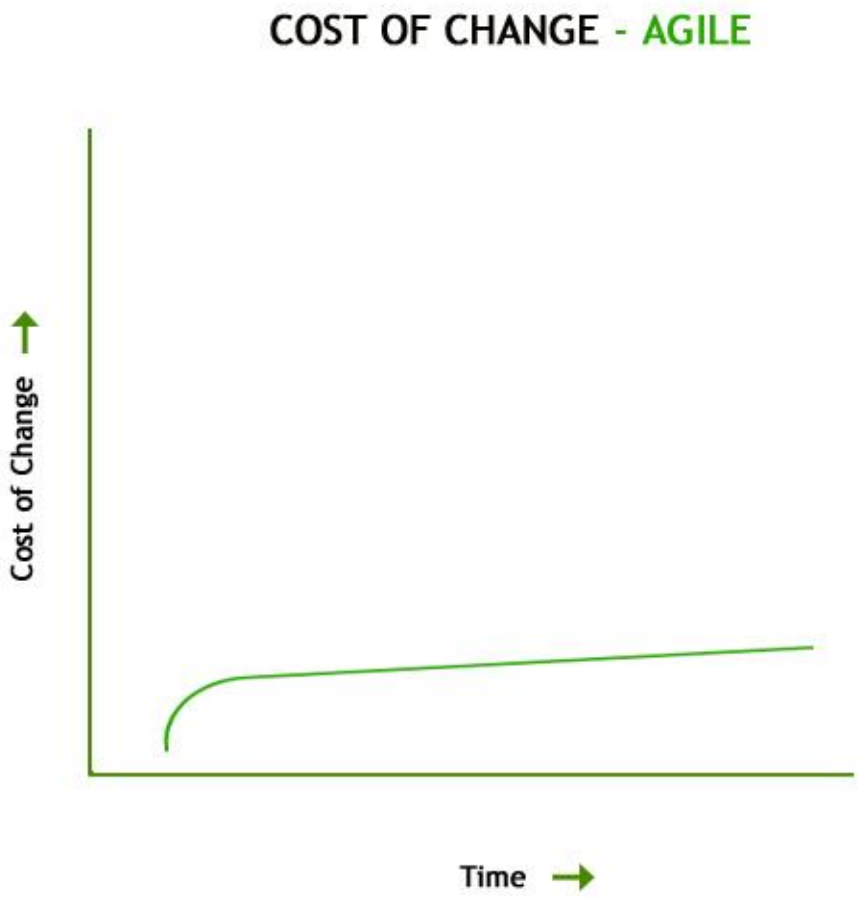


Why Agile?


What we think is the target
Waterfall, Big-Bang
Incremental
Evolutionary
The real target

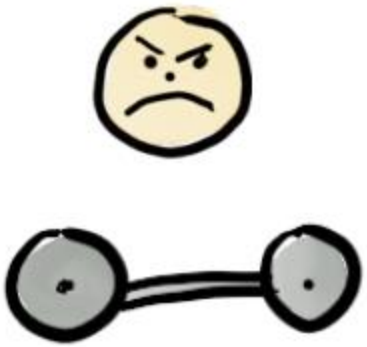
Why Agile?


COST OF CHANGE - WATERFALL
Cost of Change
Requirements
Analysis & Design
Coding
Testing in the Large
Production
Time


COST OF CHANGE - AGILE
Cost of Change
Time

Not like this....

1



2

3

4

Like this!

1

2

3

4

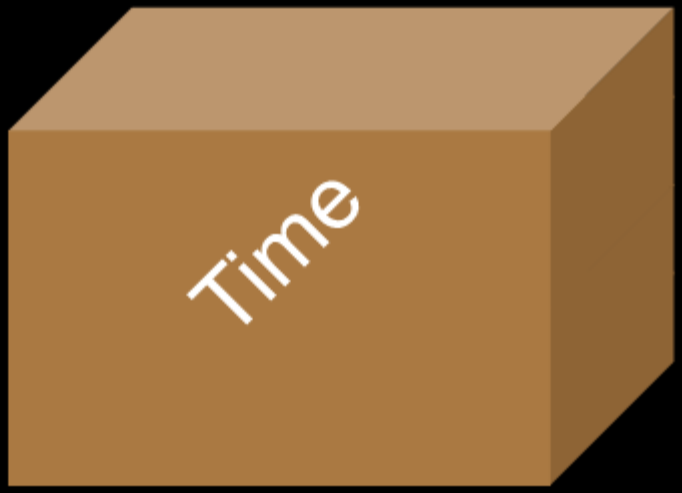
5

Maximize Value, not Output

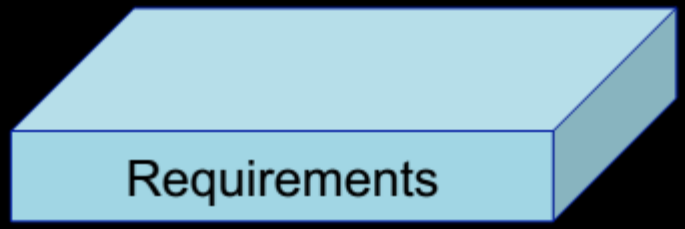
Agile vs. Waterfall

	Agile Methods	Waterfall Methods
Approach	Adaptive	Predictive
Success Measurement	Business Value	Conformation to plan
Management Style	Decentralized	Autocratic/centralized
Perspective to Change	Change Adaptability	Change Sustainability
Culture	Leadership-Collaboration	Command-Control
Documentation	Low	Heavy
Emphasis	People-Oriented	Process-Oriented
Cycles	Numerous	Limited
Upfront Planning	Minimal	Comprehensive
ROI	Early in Project	End of Project
Team Size	Small/Creative	Large

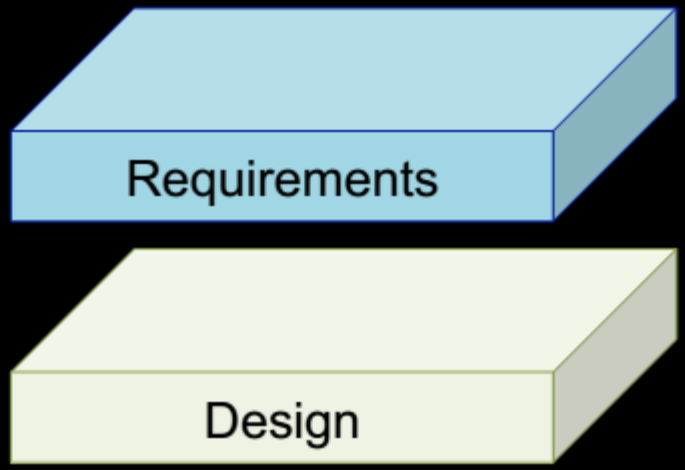
Waterfall cake



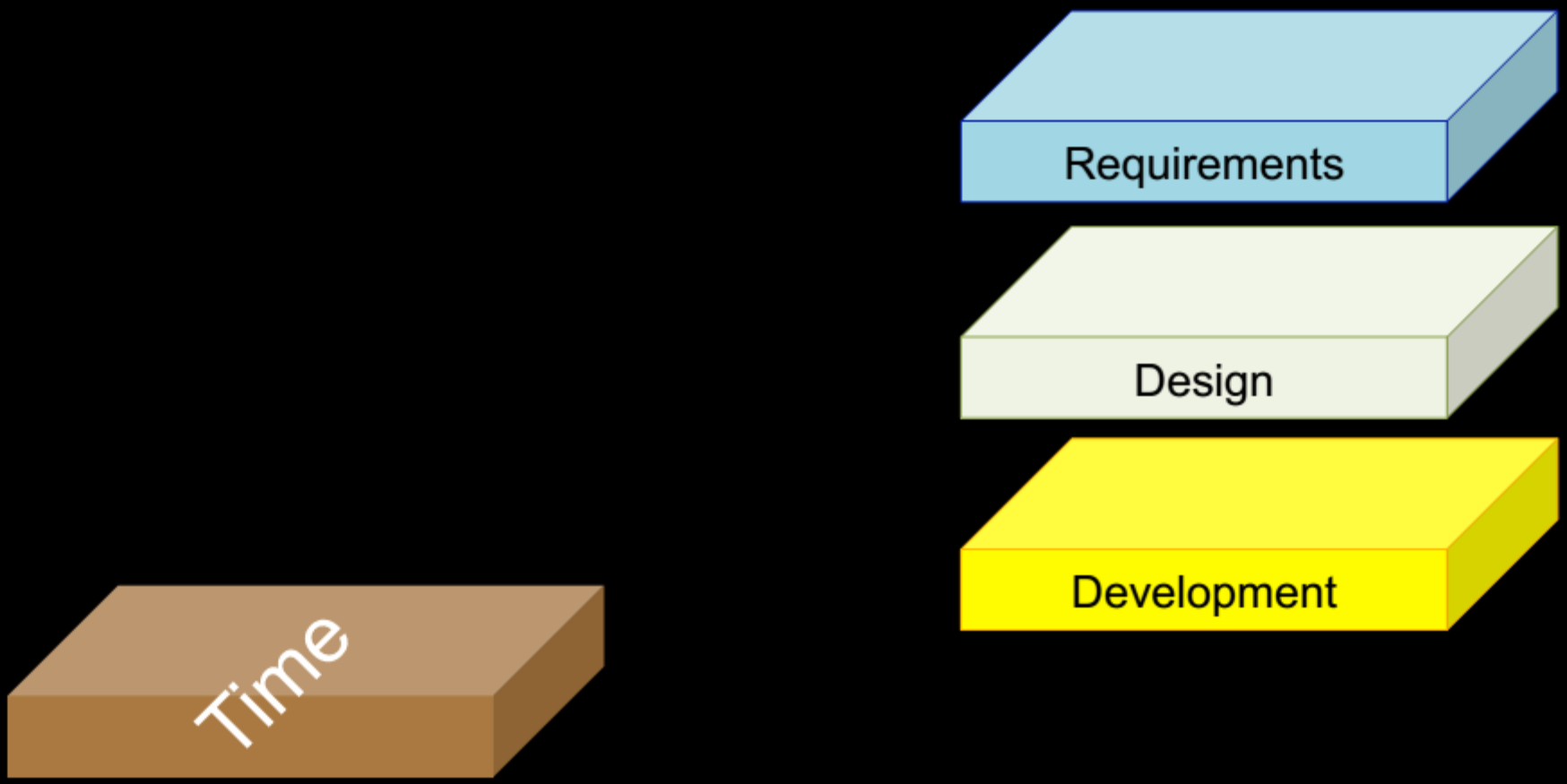
Waterfall cake



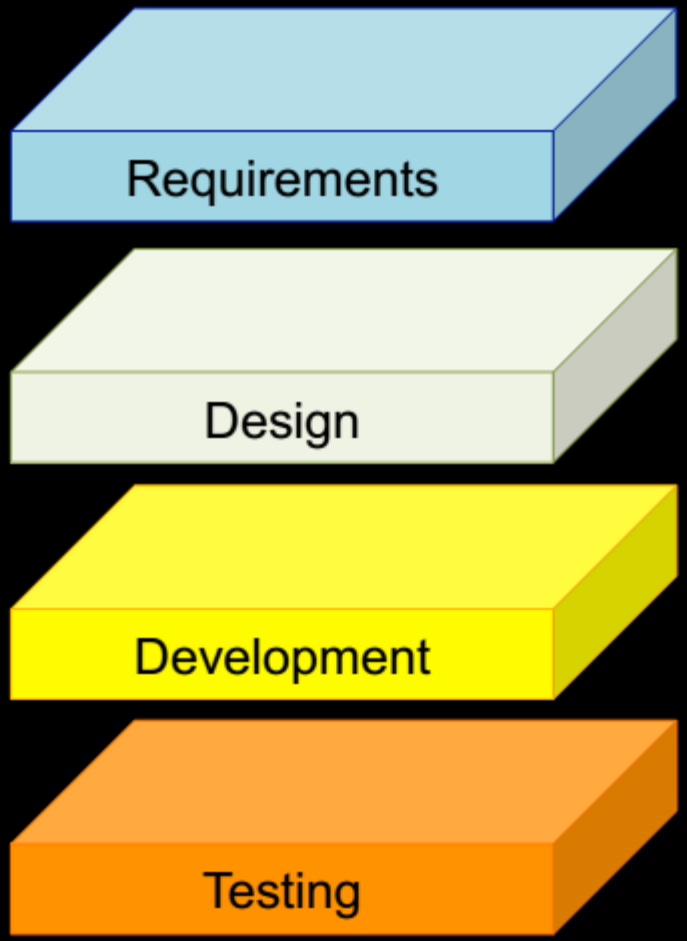
Waterfall cake



Waterfall cake

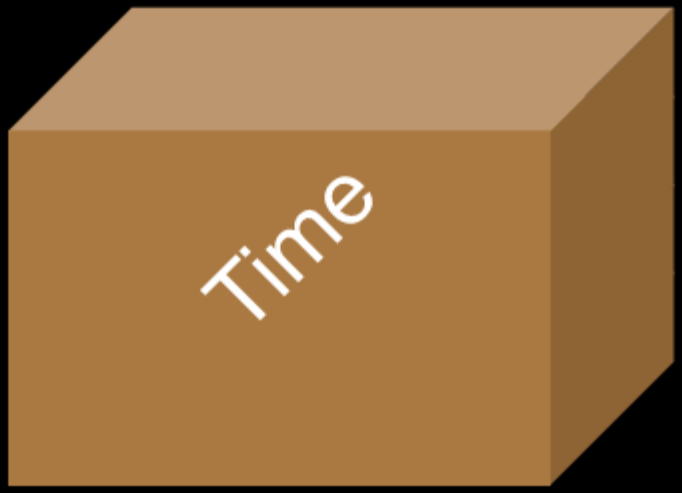


Waterfall cake

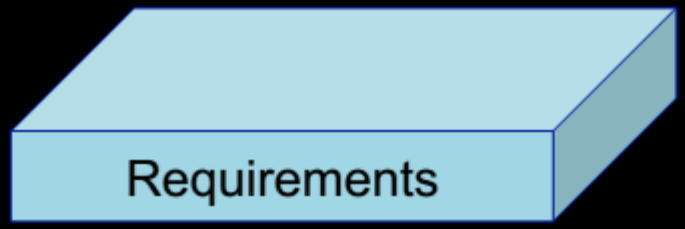


Waterfall cake

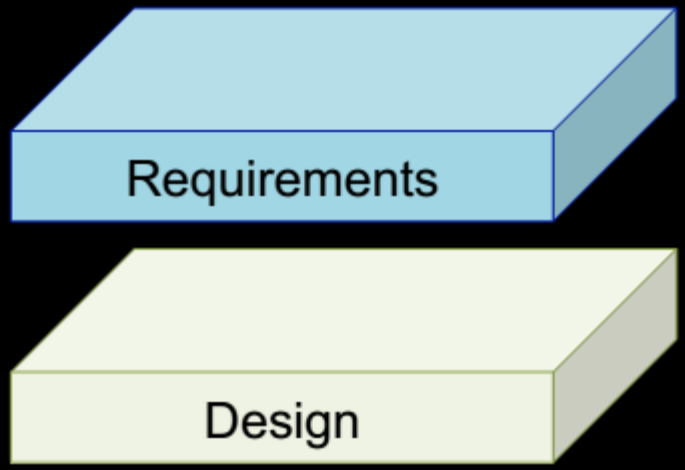
Waterfall cake



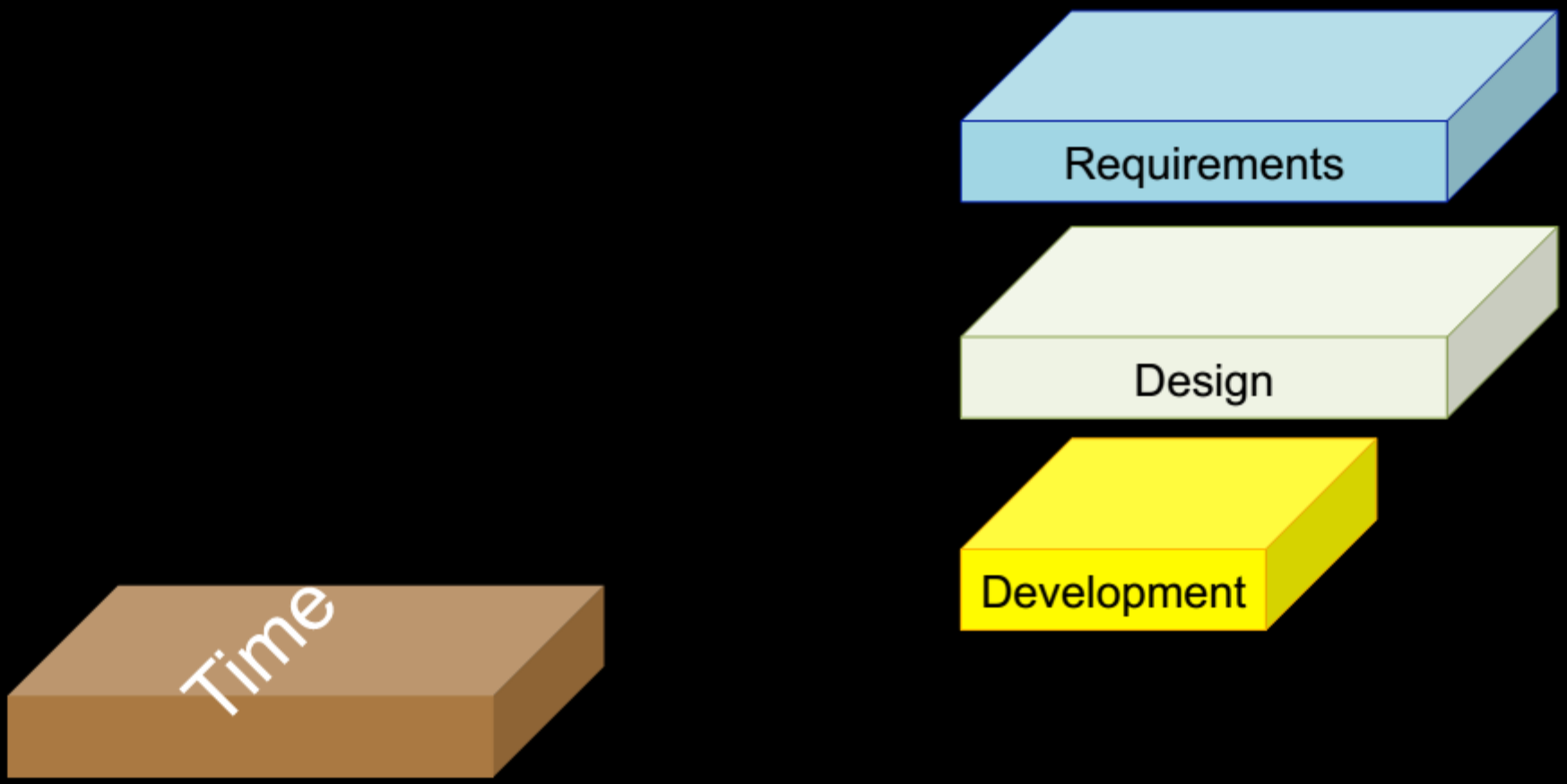
Waterfall cake



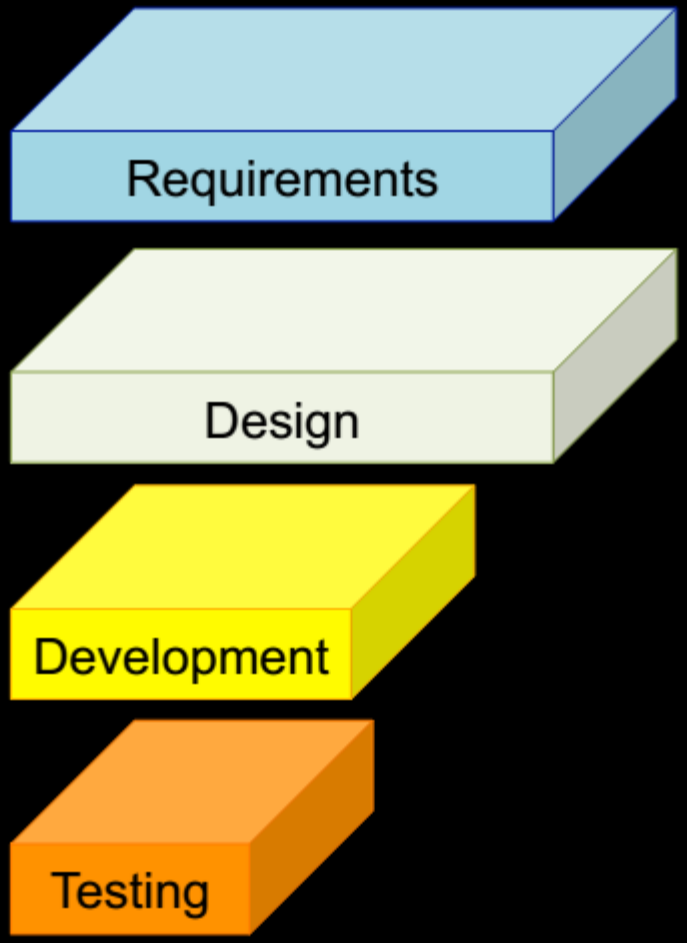
Waterfall cake



Waterfall cake



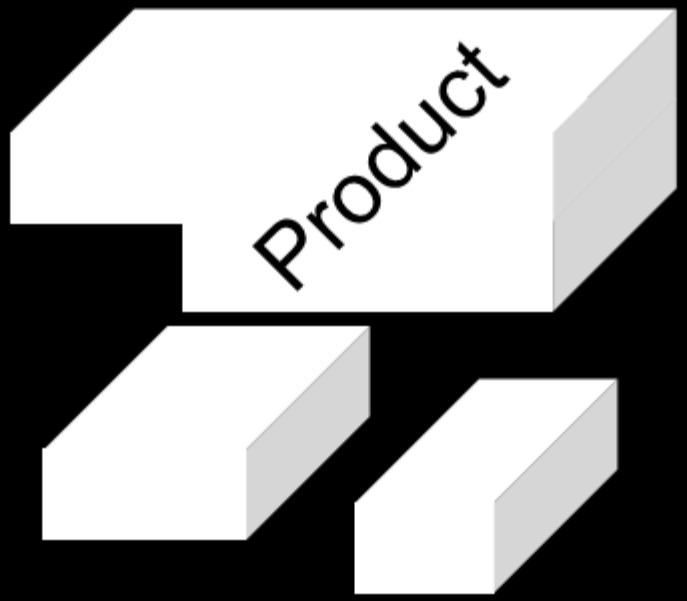
Waterfall cake



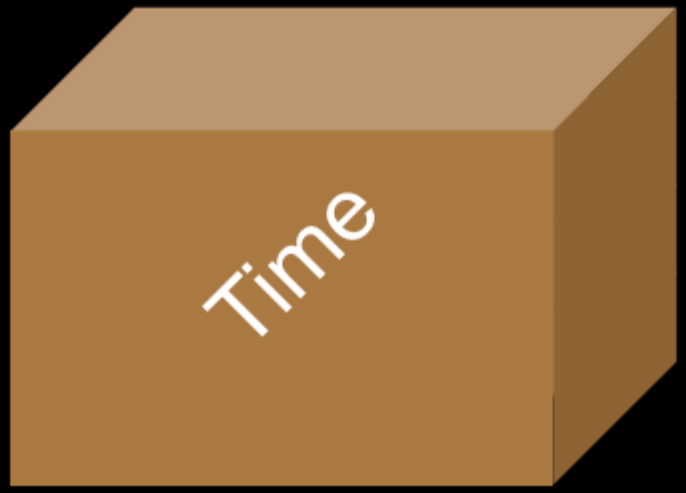
Waterfall cake

Requested

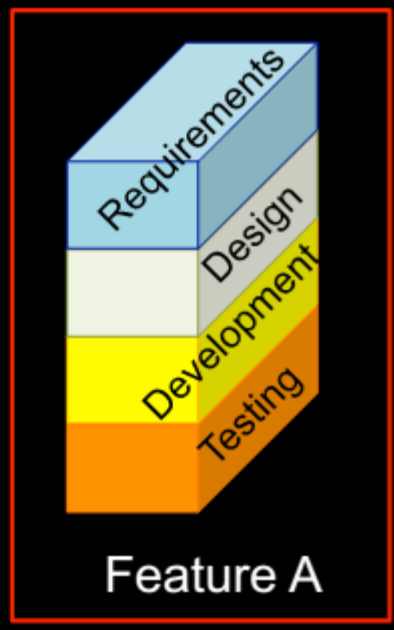
Actual



Scrum cake



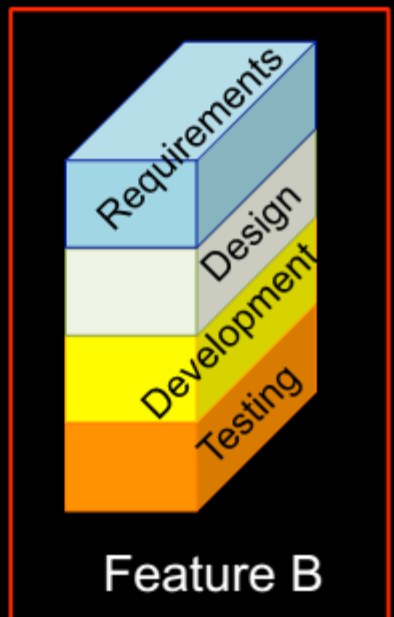
Scrum cake



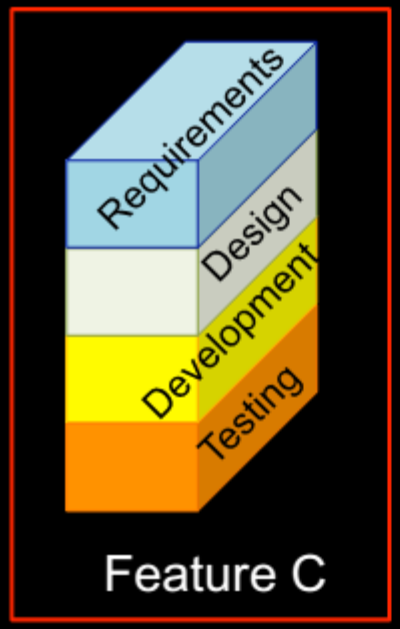
Scrum cake


Time


Feature A


Requirements
Design
Development
Testing
Feature B

Scrum cake

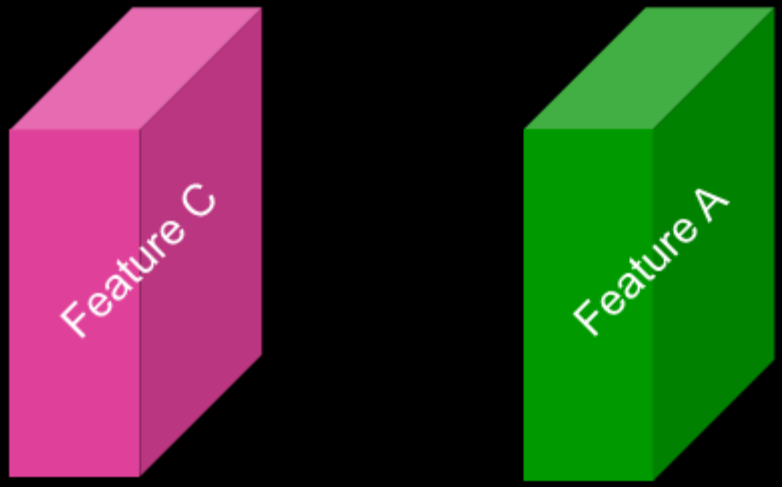

Requirements
Design
Development
Testing
Feature C

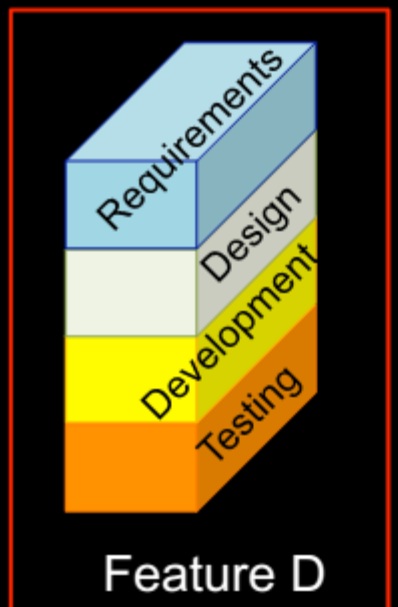

Feature A


Time


Feature B

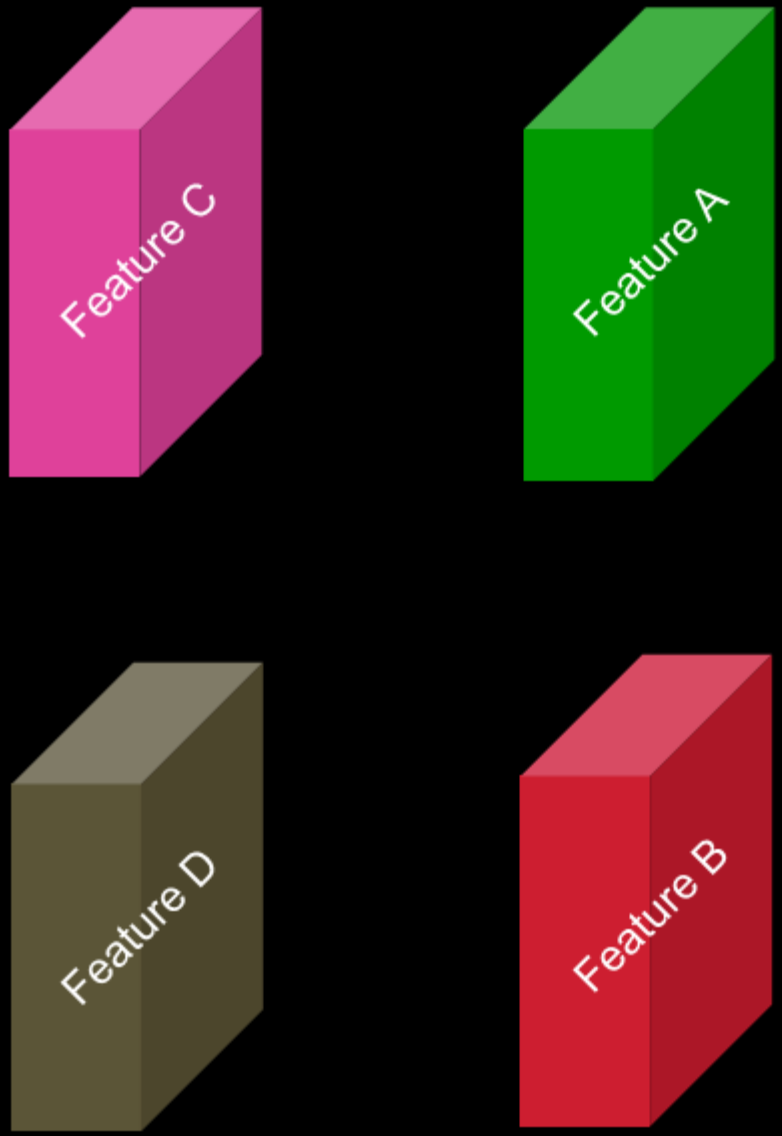
Scrum cake


Feature C
Feature A


Requirements
Design
Development
Testing
Feature D


Feature B

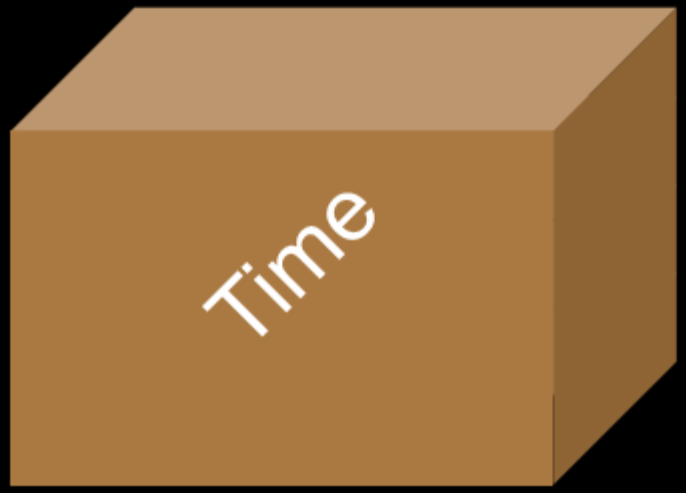
Scrum cake



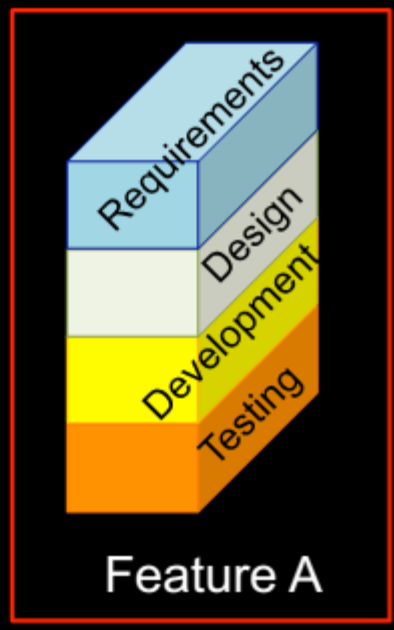
Scrum cake

Ideal

Scrum cake



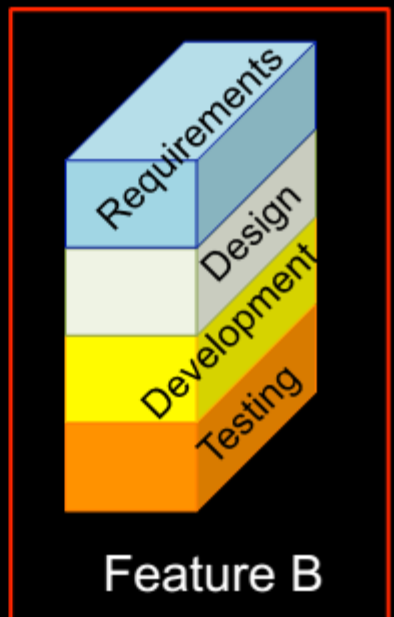
Scrum cake



Scrum cake

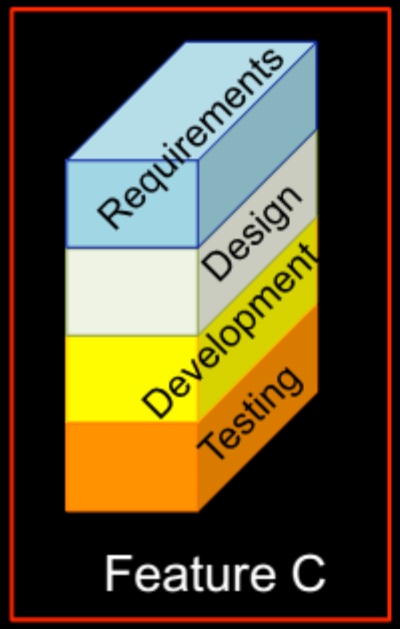

Time


Feature A


Requirements
Design
Development
Testing
Feature B

Scrum cake

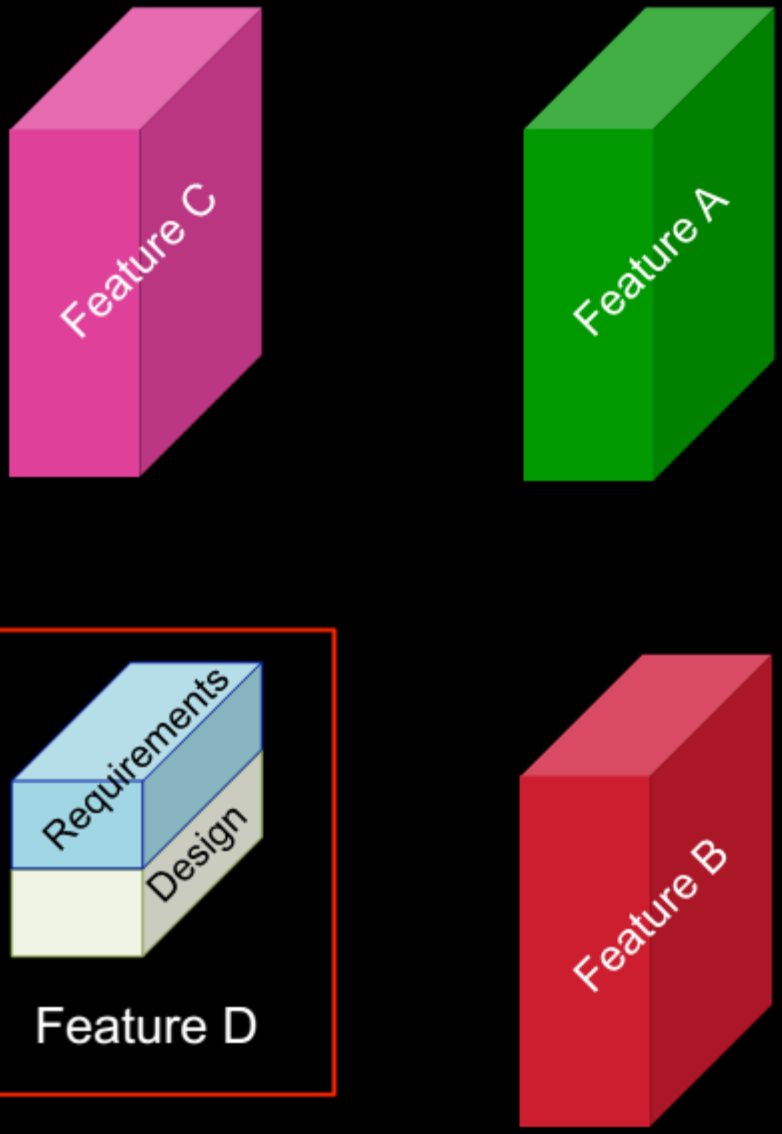

Time


Requirements
Design
Development
Testing
Feature C

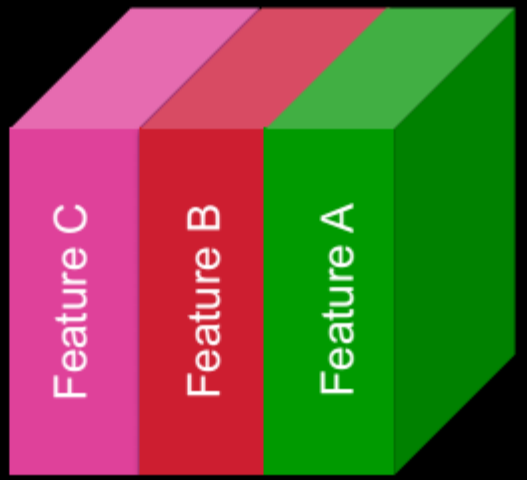

Feature A


Feature B

Scrum cake


Feature C
Feature A
Requirements
Design
Feature D
Feature B

Scrum cake



Scrum cake

Requested

Actual

Agile Methods

- Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods. These methods:
  - Focus on the code rather than the design
  - Are based on an iterative approach to software development
  - Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.
- The aim of agile methods is to reduce overheads in the software process (e.g. by limiting documentation) and to be able to respond quickly to changing requirements without excessive rework.

Agile Method Applicability

- Product development where a software company is developing a small or medium-sized product for sale.
- Custom system development within an organization, where there is a clear commitment from the customer to become involved in the development process and where there are not a lot of external rules and regulations that affect the software.
- Because of their focus on small, tightly-integrated teams, there are problems in scaling agile methods to large systems.

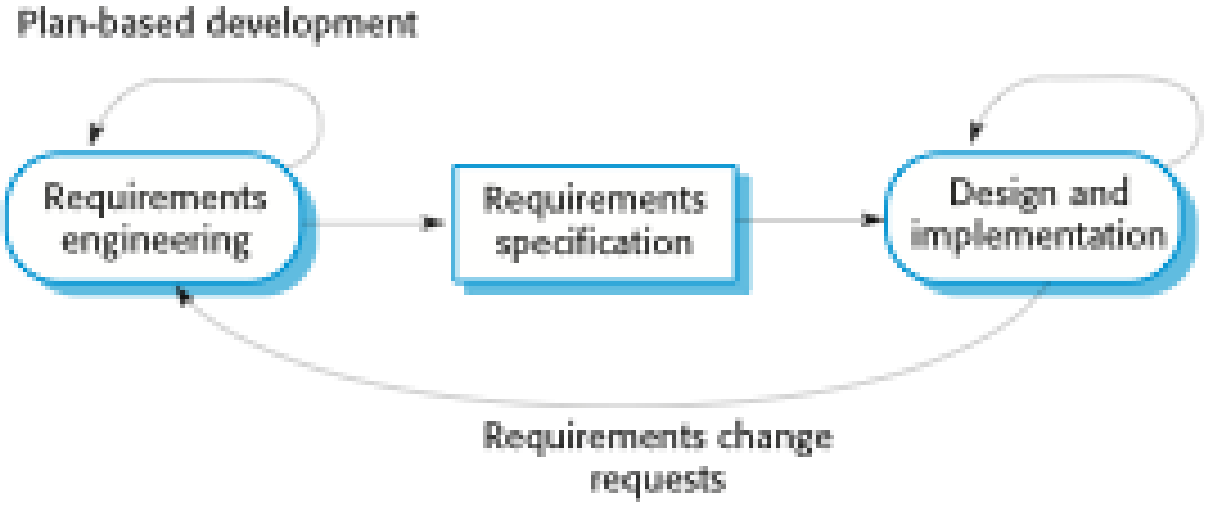
Problems With Agile Methods

- It can be difficult to keep the interest of customers who are involved in the process.
- Team members may be unsuited to the intense involvement that characterizes agile methods.
- Prioritizing changes can be difficult where there are multiple stakeholders.
- Maintaining simplicity requires extra work.
- Contracts may be a problem as with other approaches to iterative development.

Plan-driven And Agile Development

- Plan-driven development
  - A plan-driven approach to software engineering is based around separate development stages with the outputs to be produced at each of these stages planned in advance.
  - Not necessarily waterfall model – plan-driven, incremental development is possible
  - Iteration occurs within activities.
- Agile development
  - Specification, design, implementation and testing are inter-leaved and the outputs from the development process are decided through a process of negotiation during the software development process.

Plan-driven And Agile Development



Deciding on Plan-driven or Agile?

Most projects include elements of plan-driven and agile processes. Deciding on the balance depends on:

- Is it important to have a very detailed specification and design before moving to implementation? If so, you probably need to use a plan-driven approach.
- Is an incremental delivery strategy, where you deliver the software to customers and get rapid feedback from them, realistic? If so, consider using agile methods.
- How large is the system that is being developed? Agile methods are most effective when the system can be developed with a small co-located team who can communicate informally. This may not be possible for large systems that require larger development teams so a plan-driven approach may have to be used.

Deciding on Plan-driven or Agile?

- What type of system is being developed?
  - Plan-driven approaches may be required for systems that require a lot of analysis before implementation (e.g. real-time system with complex timing requirements).
- What is the expected system lifetime?
  - Long-lifetime systems may require more design documentation to communicate the original intentions of the system developers to the support team.
- What technologies are available to support system development?
  - Agile methods rely on good tools to keep track of an evolving design
- How is the development team organized?
  - If the development team is distributed or if part of the development is being outsourced, then you may need to develop design documents to communicate across the development teams.

Deciding on Plan-driven or Agile?

- Are there cultural or organizational issues that may affect the system development?
  - Traditional engineering organizations have a culture of plan-based development, as this is the norm in engineering.
- How good are the designers and programmers in the development team?
  - It is sometimes argued that agile methods require higher skill levels than plan-based approaches in which programmers simply translate a detailed design into code
- Is the system subject to external regulation?
  - If a system has to be approved by an external regulator (e.g. the FAA approve software that is critical to the operation of an aircraft) then you will probably be required to produce detailed documentation as part of the system safety case.

EXTREME PROGRAMMING (XP)

Extreme Programming at a Glance

- A type of agile software development
- Iterative approach to development
- having multiple short development cycles, rather than a long cycles
- Use of pair programming
- Writing unit tests before programming
- Simple design which constantly evolves
- Code is owned collectively

Extreme Programming

- Perhaps the best-known and most widely used agile method.
- Extreme Programming (XP) takes an 'extreme' approach to iterative development.
  - New versions may be built several times per day;
  - Increments are delivered to customers every 2 weeks;
  - All tests must be run for every build and the build is only accepted if tests run successfully.

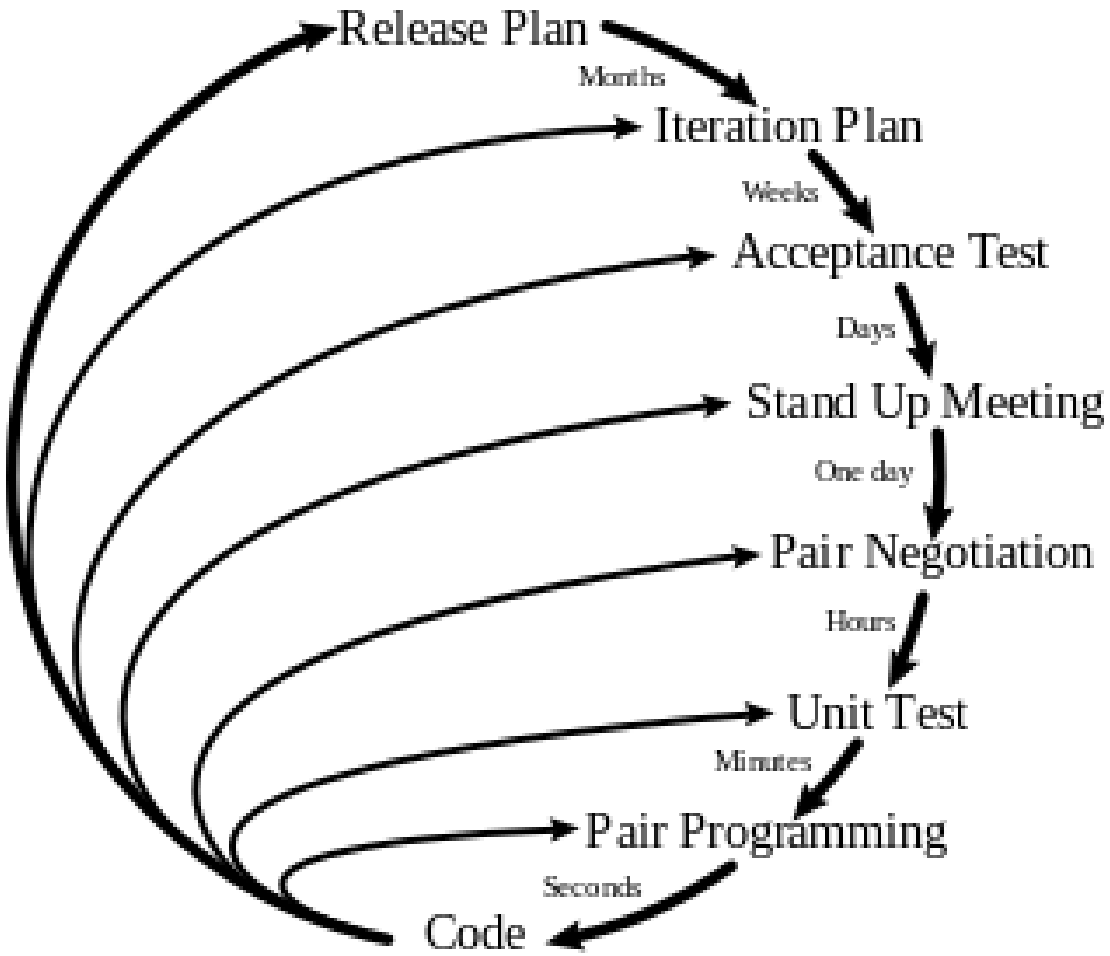
XP phases

- Exploration - customers write story cards. Project team becomes familiar with tools, technology and practices
- Planning - set priority of stories and contents first release
- Iterations to Release - iterations before releases
- Productionizing - extra testing and checking before release to customer
- Maintenance and Death - new iterations and no more customer stories

XP roles and responsibilities

- **The Developer -** writes tests and then code
- **Customer -** writes stories and functional tests
- **Tester -** helps customer write tests and runs them
- **Tracker -** keeps track on schedule, calculates progress vs. schedule on iterations
- **Coach -** person overseeing the whole process, a guide and mentor

Planning/Feedback Loops



XP Practices

- **Planning game -** programmers estimate effort of implementing customer stories and customer decides about scope and timing of releases
- **Short releases -** new release every 2-3 months
- **Simple design -** emphasis on simplest design
- **Testing -** development test driven. Unit tests before code
- **Refactoring -** restructuring and changes to simplify
- **Pair Programming -** 2 people at 1 computer

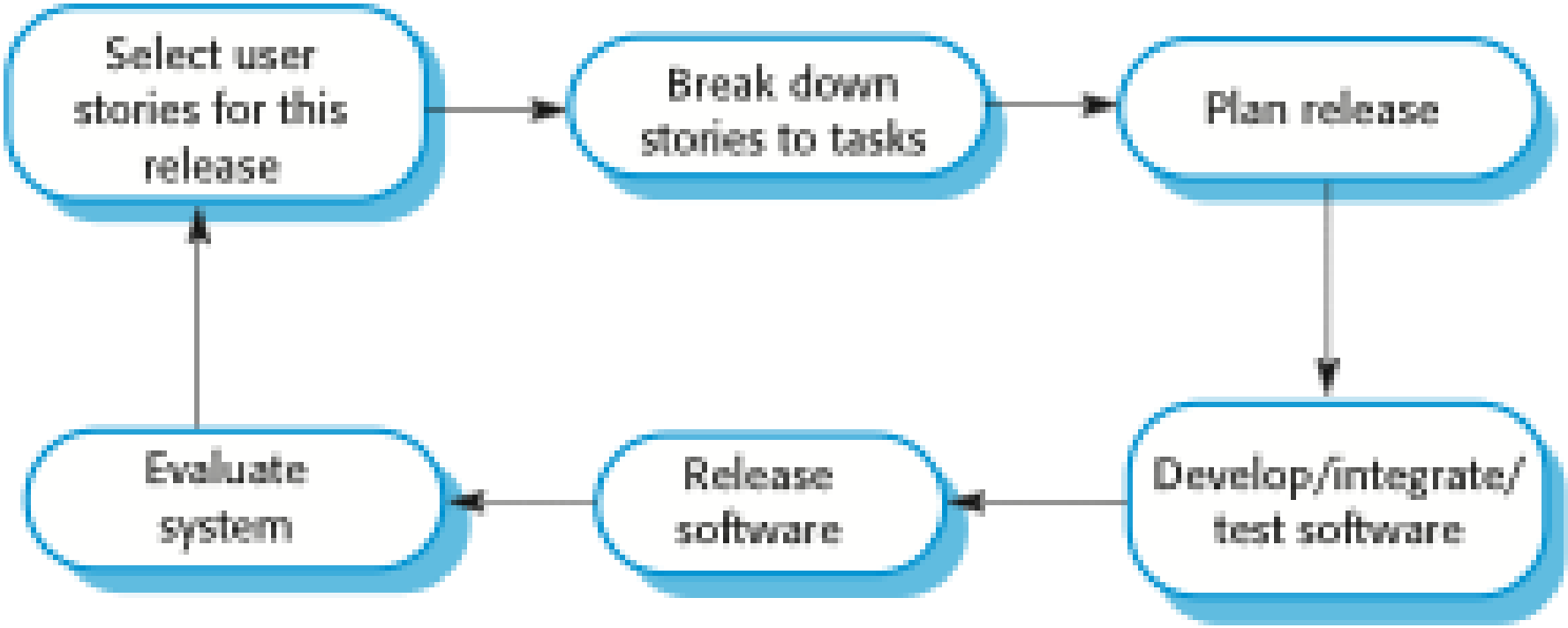
XP Practices

- **Collective ownership -** anyone can change any part of the code at any time.
- **Continuous integration -** new builds as soon as code ready
- **40 hour week -** maximum 40-hour week. No overtime
- **On-site customer -** customer present and available full-time for team
- **Coding standards -** rules exist and are followed
- **Open workspace -** large room small cubicles

XP and Agile Principles

- Incremental development is supported through small, frequent system releases.
- Customer involvement means full-time customer engagement with the team.
- People not process through pair programming, collective ownership and a process that avoids long working hours.
- Change supported through regular system releases.
- Maintaining simplicity through constant refactoring of code.

The Extreme Programming Release Cycle



Extreme Programming Practices

Principle or practice	Description
Incremental planning	Requirements are recorded on story cards and the stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development 'Tasks'.
Small releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Simple design	Enough design is carried out to meet the current requirements and no more.
Test-first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.

Extreme Programming Practices

Pair programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
Collective ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all of the code. Anyone can change anything.
Continuous integration	As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium term productivity
On-site customer	A representative of the end-user of the system (the customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

Requirements Scenarios (Customer Stories)

- In XP, a customer or user is part of the XP team and is responsible for making decisions on requirements.
- User requirements are expressed as scenarios or user stories.
- These are written on cards and the development team break them down into implementation tasks. These tasks are the basis of schedule and cost estimates.
- The customer chooses the stories for inclusion in the next release based on their priorities and the schedule estimates.

Customer Involvement

- The role of the customer in the testing process is to help develop acceptance tests for the stories that are to be implemented in the next release of the system.
- The customer who is part of the team writes tests as development proceeds. All new code is therefore validated to ensure that it is what the customer needs.
- However, people adopting the customer role have limited time available and so cannot work full-time with the development team. They may feel that providing the requirements was enough of a contribution and so may be reluctant to get involved in the testing process.

XP and Change

- Conventional wisdom in software engineering is to design for change. It is worth spending time and effort anticipating changes as this reduces costs later in the life cycle.
- XP, however, maintains that this is not worthwhile as changes cannot be reliably anticipated.
- Rather, it proposes constant code improvement (refactoring) to make changes easier when they have to be implemented.

Refactoring

- Programming team look for possible software improvements and make these improvements even where there is no immediate need for them.
- This improves the understandability of the software and so reduces the need for documentation.
- Changes are easier to make because the code is well-structured and clear.
- However, some changes requires architecture refactoring and this is much more expensive.

Examples Of Refactoring

- Re-organization of a class hierarchy to remove duplicate code.
- Tidying up and renaming attributes and methods to make them easier to understand.
- The replacement of inline code with calls to methods that have been included in a program library.

Testing in XP

- Testing is central to XP and XP has developed an approach where the program is tested after every change has been made.
- XP testing features:
  - Test-first development.
  - Incremental test development from scenarios.
  - User involvement in test development and validation.
  - Automated tests are used to run all component tests each time that a new release is built.

Test-first Development

- Writing tests before code clarifies the requirements to be implemented.
- Tests are written as programs rather than data so that they can be executed automatically. The test includes a check that it has executed correctly.
  - Usually relies on a testing framework such as *Junit*.
- All previous and new tests are run automatically when new functionality is added, thus checking that the new functionality has not introduced errors.

XP Testing Difficulties

- Programmers prefer programming to testing and sometimes they take short cuts when writing tests. For example, they may write incomplete tests that do not check for all possible exceptions that may occur.

- It difficult to judge the completeness of a set of tests. Although you may have a lot of system tests, your test set may not provide complete coverage.

Pair Programming

- In XP, programmers work in pairs, sitting together to develop code.
- This helps develop common ownership of code and spreads knowledge across the team.
- It serves as an informal review process as each line of code is looked at by more than 1 person.
- It encourages refactoring as the whole team can benefit from this.

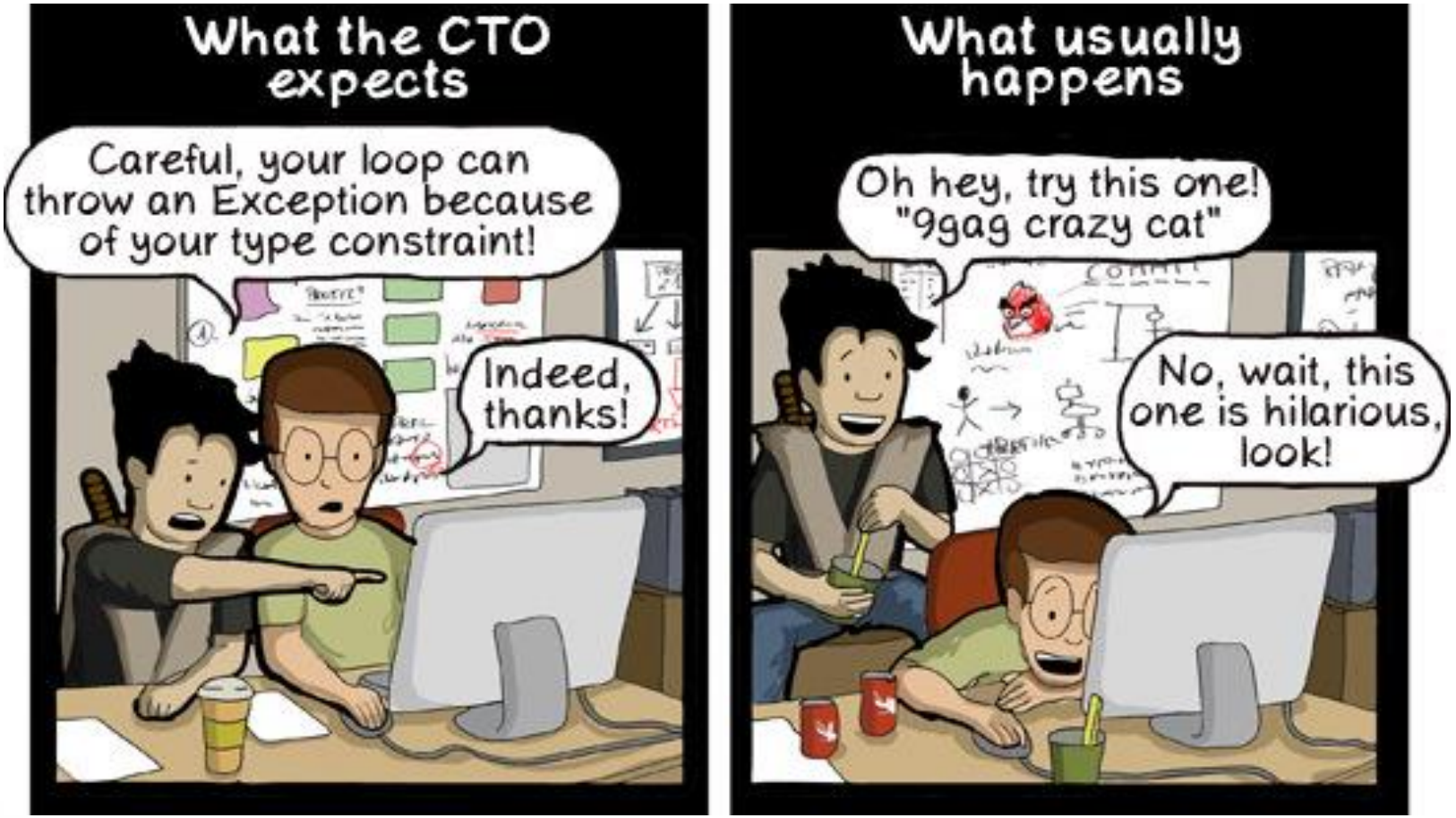
Pair Programming

- In pair programming, programmers sit together at the same workstation to develop the software.
- Pairs are created dynamically so that all team members work with each other during the development process.
- The sharing of knowledge that happens during pair programming is very important as it reduces the overall risks to a project when team members leave.
- Pair programming is not necessarily inefficient and there is evidence that a pair working together is more efficient than 2 programmers working separately.

Advantages Of Pair Programming

- It supports the idea of collective ownership and responsibility for the system.
  - Individuals are not held responsible for problems with the code. Instead, the team has collective responsibility for resolving these problems.
- It acts as an informal review process because each line of code is looked at by at least two people.
- It helps support refactoring, which is a process of software improvement.
  - Where pair programming and collective ownership are used, others benefit immediately from the refactoring so they are likely to support the process.

Pair Programming



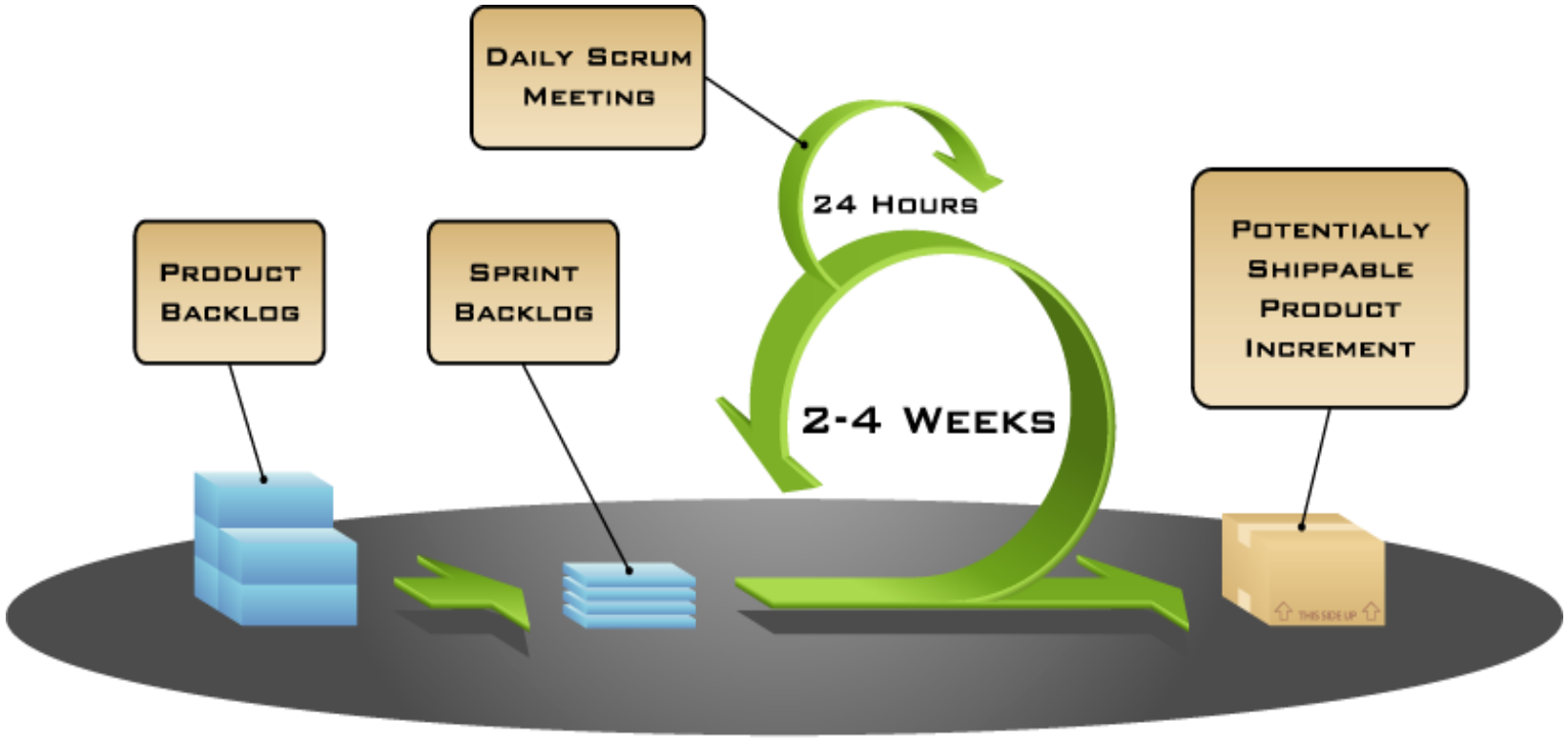
SCRUM

Scrum is…

"An agile framework that allows us to focus on delivering the highest business value in the shortest time."

– Mike Cohn – "Introduction to Scrum Methodology"

Scrum: Structure



Overview of SCRUM

- Initiating:
  - Determine roles
  - Decide how many sprints will compose each release and the scope of software to deliver
- Planning:
  - Create product backlog
  - Create sprint backlog
  - Create release backlog
  - Plan work each day in the daily Scrum
  - Document stumbling blocks in a list
- Executing:
  - Complete tasks each day during sprints
  - Produce a shippable product at the end of each sprint
- Monitoring and Controlling:
  - Resolve issues and blockers
  - Create and update burndown chart
  - Demonstrate the completed product during the sprint review meeting
- Closing:
  - Reflect on how to improve the product and process during the sprint reflection meeting

Sprint 0

This stage focuses on understanding the project in terms of business and technical requirements and setting the project governance along with the initial project management activities.

Key Activities

- Conduct workshops to understand/clarify requirements
- Document, review and refine requirements. This will generate the product backlog.
- Estimate the effort required for the phase (high-level, ball park estimate)
- Required planning activities are performed for the entire phase

Key Deliverables

- Software Requirements Specification (updated with use cases)
- High-level Architecture & Design Document
- Deployment Model
- Project Schedule (high-level milestones, etc)
- Quality Assurance Test Strategy / Plan

The Sprint cycle

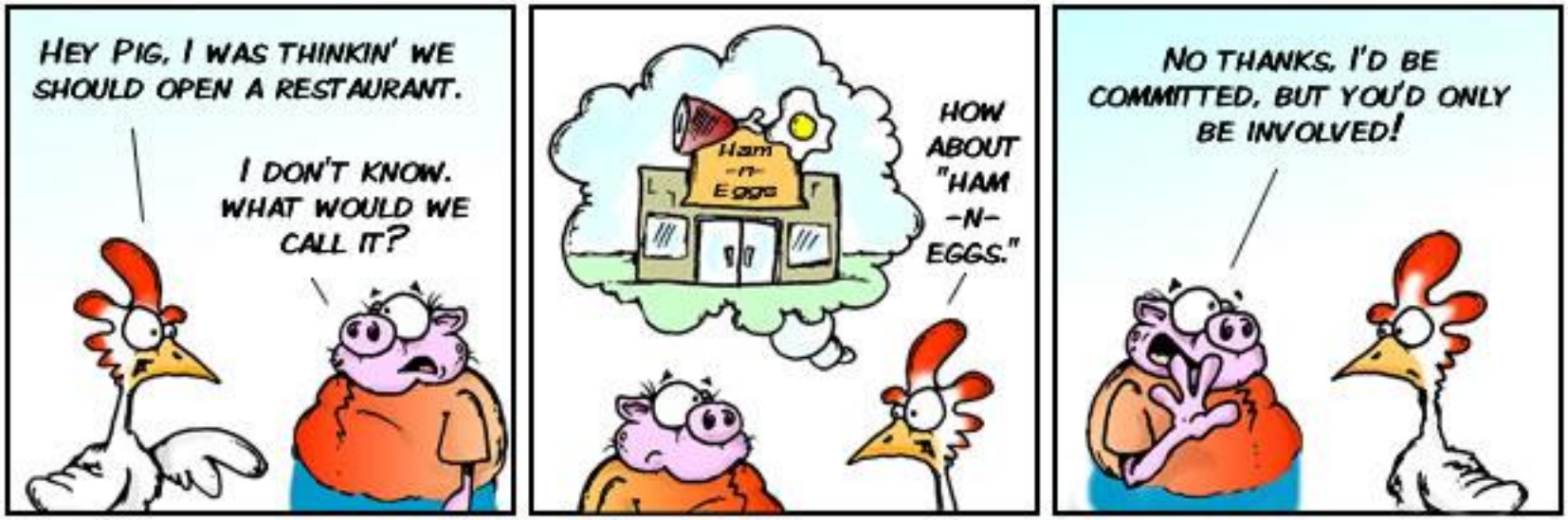
- Sprints are fixed length, normally 2–4 weeks. They correspond to the development of a release of the system in XP.
- The starting point for planning is the product backlog, which is the list of work to be done on the project.
- The selection phase involves all of the project team who work with the customer to select the features and functionality to be developed during the sprint.

The Sprint cycle

- Once these are agreed, the team organize themselves to develop the software. During this stage the team is isolated from the customer and the organization, with all communications channelled through the so-called 'Scrum master'.
- The role of the Scrum master is to protect the development team from external distractions.
- At the end of the sprint, the work done is reviewed and presented to stakeholders. The next sprint cycle then begins.

Teamwork in SCRUM

- The ‘Scrum master’ is a facilitator who arranges daily meetings, tracks the backlog of work to be done, records decisions, measures progress against the backlog and communicates with customers and management outside of the team.
- The whole team attends short daily meetings where all team members share information, describe their progress since the last meeting, problems that have arisen and what is planned for the following day.
  - This means that everyone on the team knows what is going on and, if problems arise, can re-plan short-term work to cope with them.


HEY PIG, I WAS THINKIN' WE SHOULD OPEN A RESTAURANT.
I DON'T KNOW. WHAT WOULD WE CALL IT?
Ham -n- Eggs
HOW ABOUT "HAM -N- EGGS."
NO THANKS. I'D BE COMMITTED, BUT YOU'D ONLY BE INVOLVED!
By Clark & Vizdos
© 2006 implementingscrum.com

Scrum – Roles & Responsibilities

Roles:

- **Scrum Master** - who maintains the processes (equivalent to Project Manager)
- **Product Owner** - who represents the stakeholders
- **Team** - a cross-functional group

Scrum consists of a series of Sprints

- Each Sprint involves:
  - Sprint planning
  - Daily Activities
  - Sprint Demo
  - Sprint Retrospective

- Each Sprint is a duration of 2 to 4 weeks (generally)

Scrum Artefacts

In Scrum, an artifact is a useful object created by people. An artifact can be called a deliverable in other project management approaches. The following three artifacts are created with Scrum:

Scrum Artefacts

- **Product backlog**:
  - A single list of features prioritized by business value.
  - The highest priority items should be broken down in enough detail for the team to estimate the effort involved in developing them.
  - Some experts suggest scheduling about ten work days for each item
- **Sprint backlog**:
  - The highest-priority items from the product backlog to be completed within a sprint.
  - The Scrum team breaks down the highest-priority items into smaller tasks that take about 16 hours to complete.
- **Burndown chart**:
  - Shows the cumulative work remaining in a sprint on a day-by-day basis.

Sprint Cycles

This stage focuses on building up the product in an incremental manner. During each sprint, the team creates an increment of potential shippable software. The set of features that go into each sprint come from the product backlog, which is a prioritized set of high level requirements of work to be done.

Sprint Cycles - Key Activities

- Sprint Planning
  - Definition of a new sprint based on currently known backlog. Sprint backlog will be created by determining which items on the backlog move in to the sprint.
- Daily Activities
  - Design: Design how the backlog items will be implemented. This includes system architecture modification and high level design
  - Implement: Development of new release functionalities as per the requirements and the design
  - Test: Functional testing of the features developed in the particular sprint
  - Daily stand-ups: Daily meetings with onsite and offshore Scrum teams (max 20 mins), to discuss ‘what was completed yesterday’, ‘what is to be completed today’
- Sprint Review
  - A walkthrough of all the functionalities developed in that particular sprint in order to collect client feedback.
- Sprint Retrospective
  - A meeting where all team members (including the client) reflect about the past sprint; especially on ‘what went right?’, ‘what went wrong?’ and ‘what could be improved in the next sprint?’

Sprint Cycles – Key Deliverables

- Working and tested software (as per the sprint backlog)
- Sprint Release Notes
- Test Report

Scrum - Managing the Change

During a Sprint:

- Once team has committed, no changes to Sprint scope
- No Changes to Deliverable
- Details will emerge during Sprint, but no new work or substantially changed work
- Customer can terminate the Sprint if necessary
- No Changes to Sprint Duration
- Sprint ends on planned date whether team has completed its commitment or not

Customer can make any changes to the remaining Product Backlog before the start of the next Sprint

SCRUM Benefits

- The product is broken down into a set of manageable and understandable chunks.
- Unstable requirements do not hold up progress.
- The whole team have visibility of everything and consequently team communication is improved.
- Customers see on-time delivery of increments and gain feedback on how the product works.
- Trust between customers and developers is established and a positive culture is created in which everyone expects the project to succeed.

Agile Adoption Challenges

- It's hard to break the waterfall mind-set
- This is a continuous process, not a one-off change
- Communication requirements are higher in agile processes
- This is a radical change
    - Confusion & pain is inevitable
- Maintaining simplicity requires extra work
- Keeping client stakeholders engaged on short frequent cycles throughout the project

Agile Adoption Challenges

- Require good mediators/coordinators
  - To solve communication issues due to language and cultural differences
  - To suggest improvements continuously
  - To act as “harmonizers”
- Lots of discipline
- All team members may be not be suitable to the intense involvement that characterizes Agile methods
- Multiple stakeholders make requirements and change management complex

Scaling Agile Methods

- Agile methods have proved to be successful for small and medium sized projects that can be developed by a small co-located team.
- It is sometimes argued that the success of these methods comes because of improved communications which is possible when everyone is working together.
- Scaling up agile methods involves changing these to cope with larger, longer projects where there are multiple development teams, perhaps working in different locations.

Large Systems Development

- Large systems are usually collections of separate, communicating systems, where separate teams develop each system. Frequently, these teams are working in different places, sometimes in different time zones.
- Large systems include and interact with a number of existing systems. Many of the system requirements are concerned with this interaction and so don't really lend themselves to flexibility and incremental development.
- Where several systems are integrated to create a system, a significant fraction of the development is concerned with system configuration rather than original code development.

Large Systems Development

- Large systems and their development processes are often constrained by external rules and regulations limiting the way that they can be developed.
- Large systems have a long procurement and development time. It is difficult to maintain coherent teams who know about the system over that period as, inevitably, people move on to other jobs and projects.
- Large systems usually have a diverse set of stakeholders. It is practically impossible to involve all of these different stakeholders in the development process.

Scaling Out And Scaling Up

- 'Scaling up' is concerned with using agile methods for developing large software systems that cannot be developed by a small team.
- 'Scaling out' is concerned with how agile methods can be introduced across a large organization with many years of software development experience.
- When scaling agile methods it is essential to maintain agile fundamentals
  - Flexible planning, frequent system releases, continuous integration, test-driven development and good team communications.

Scaling Up To Large Systems

- For large systems development, it is not possible to focus only on the code of the system. You need to do more up-front design and system documentation
- Cross-team communication mechanisms have to be designed and used. This should involve regular phone and video conferences between team members and frequent, short electronic meetings where teams update each other on progress.
- Continuous integration, where the whole system is built every time any developer checks in a change, is practically impossible. However, it is essential to maintain frequent system builds and regular releases of the system.

Scaling Out To Large Companies

- Project managers who do not have experience of agile methods may be reluctant to accept the risk of a new approach.
- Large organizations often have quality procedures and standards that all projects are expected to follow and, because of their bureaucratic nature, these are likely to be incompatible with agile methods.
- Agile methods seem to work best when team members have a relatively high skill level. However, within large organizations, there are likely to be a wide range of skills and abilities.
- There may be cultural resistance to agile methods, especially in those organizations that have a long history of using conventional systems engineering processes.

Key Points

- Agile methods are incremental development methods that focus on rapid development, frequent releases of the software, reducing process overheads and producing high-quality code. They involve the customer directly in the development process.
- The decision on whether to use an agile or a plan-driven approach to development should depend on the type of software being developed, the capabilities of the development team and the culture of the company developing the system.
- Extreme programming is a well-known agile method that integrates a range of good programming practices such as frequent releases of the software, continuous software improvement and customer participation in the development team.

Key Points

- A particular strength of extreme programming is the development of automated tests before a program feature is created. All tests must successfully execute when an increment is integrated into a system.
- The Scrum method is an agile method that provides a project management framework. It is centred round a set of sprints, which are fixed time periods when a system increment is developed.
- Scaling agile methods for large systems is difficult. Large systems need up-front design and some documentation.

Other Process Models

- Component based development—the process to apply when reuse is a development objective
- Formal methods—emphasizes the mathematical specification of requirements
- Aspect-Oriented Software Development — Design and development of parts of a program that rely on or must affect many other parts of the system. They form the basis for the development of aspects.

COMPUTER AIDED SOFTWARE ENGINEERING (CASE)

Overview of CASE Approach

- Computer Aided Software Engineering (CASE) is the use of computer-based support in the software development process.
- Embedding CASE technology in software process led improvements in software quality and productivity.

CASE Automated Activities

- The development of graphical system models
- Understanding a design using a data dictionary
- Generation of user interfaces
- Program debugging
- The automated translation of programs from an old version

CASE Limited Factors

- Providing artificial intelligence technology to support design activities will be unsuccessful.

- Software engineering is a team based activity that require interactivity among team members. CASE technology does not support for this.

CASE Tools

- Tools used to assist the all aspects of the software development lifecycle are known as CASE Tools.

- Some typical CASE tools are:
  - Code generation tools
  - Data modeling tools
  - UML
  - Refactoring tools
  - Model transformation Tools
  - Configuration management tools including revision control

CASE Classification

- Help to understand the types of CASE tools and their role in supporting software process activities.

- CASE tools can be classified in three perspectives.
  1. ***Functional perspective:*** Classify according to the specific function.
  2. ***Process perspective:*** Classify according to the process activities that they support.
  3. ***Integration perspective:*** Classify according to how they are organized into integrated units that provide support for one or more process activities.

Functional Classification of CASE Tools

Tool Type	Examples
Planning Tools	PERT tools, ESTIMATION tools, Spreadsheets
Editing Tools	Text editors, diagram editors, word processors
Change Management Tools	Requirements traceability tools, change control systems
Configuration Management Tools	Version management systems, system building tools
Prototyping Tools	Very high level languages, User interface generators.
Method Support Tools	Design editors, data dictionaries, Code generators
Language-Processing Tools	Compilers, Interpreters
Program analysis Tools	Cross reference generators, Static analyzers, Dynamic analyzers
Testing Tools	Test data generators, File comparators
Debugging Tools	Interactive debugging systems
Documentation Tools	Page layout programs, Image editors
Re-engineering Tools	Cross reference systems, Program restructuring systems

Another Classification Dimension

- CASE tools can be classified according to the support offered to software process.

  - ***Tools :*** Support individual process tasks such as checking consistency of a design etc.
  - ***Workbenches :*** Support process phases or activities such as specification etc.
  - ***Environments :*** Support all or substantial part of the software process

Tools Workbenches and Environments