Software Development Life Cycle

Lecture 3: Process Models and Agile Development

DR. YOUNG SAENG PARK

2021/2022

WMG
THE UNIVERSITY OF WARWICK

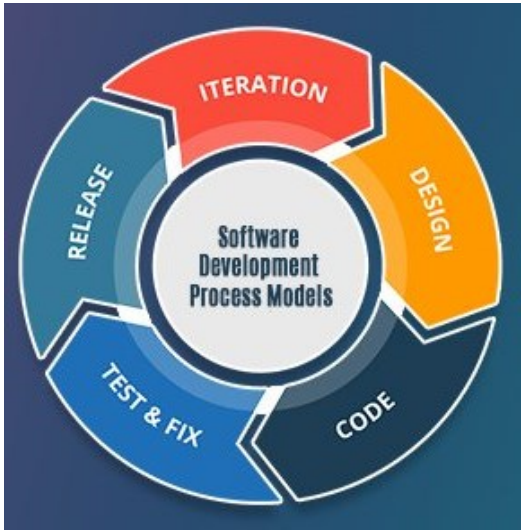
Contents

Process Models and Agile Development

SDLC Models

Software Development Life Cycle

- **Software Development Life Cycle (SDLC)** is the process of dividing software development work into **smaller**, **parallel** or **sequential steps** to improve design, product management, and project management.
- Software Development Life Cycle (SDLC) model is an **abstract representation of software process**.
- SDLC model is sometimes called **a software development process model** (simply software process model) or **software process development paradigm**.
- There are typically two types of software development process models: **plan-driven models** and **agile models**. Some example in those models are **Waterfall Model**, **Rapid Prototyping Model**, **Spiral Model**, **Scrum Model**, **XP Model** etc.
- Based on **the project's needs and preferences**, some SDLC models will be better suited than others.



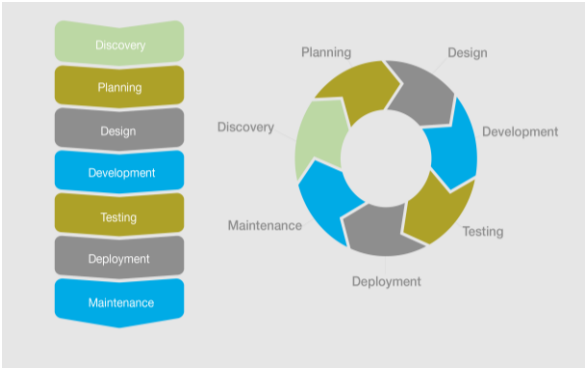
Source: Eduonix Blog

Benefits of Software Development Life Cycle

- With the SDLC, it is possible to clearly **see the goals and the problems** so that the plan is implemented with **precision** and **relevance**.
- A formal review is created at the end of each stage so that it allows to **have maximum management control**.
- The installation using the SDLC **has the necessary checks and balances** so that it will be tested with **precision before the installation**.
- The SDLC **provides a well-structured and well-documented paper** trail of the entire project with the records of everything that occurs.
- With a well-designed SDLC, **everything will be in order** so that a new project member can continue the process without complications.
- The SDLC makes **sticking to a project budget easier** with a well-organised plan so that it can see the clear **project timetables** and **costs**.
- The SDLC model **provides the project with flexibility** by having feedback into the earlier stages.

Plan-driven and Agile Development

- **Plan-driven development** are processes where all of the process activities are **planned in advance** and progress is measured against this plan.
- **Agile development** are planning based on an **incremental approach** which is easier to reflect customer requirement changes.
- Plan-driven development is **good for stable, but expensive for dynamic environments**, whereas agile development is **good for dynamic but expensive for stable environments**.
- Plan-driven development is suitable **for large systems and teams** whereas agile development is suitable **for medium systems and teams**.
- Most practical developments include **elements of both plan-driven and agile approaches**. There are no right or wrong software development processes.



Source: Novacura Flow Blog

Plan-driven and Agile Development Models

Plan-driven Development Models

- Waterfall Model
- V Model
- Spiral Model
- Incremental Model
- Iterative Model

...

Agile Development Models

- Scrum Model
- Extreme Programming (XP)
- Kanban
- Feature Driven Development
- Rapid Application Development (RAD)
- Dynamic System Development Method

...

Process Models and Agile Development

Plan-driven Models

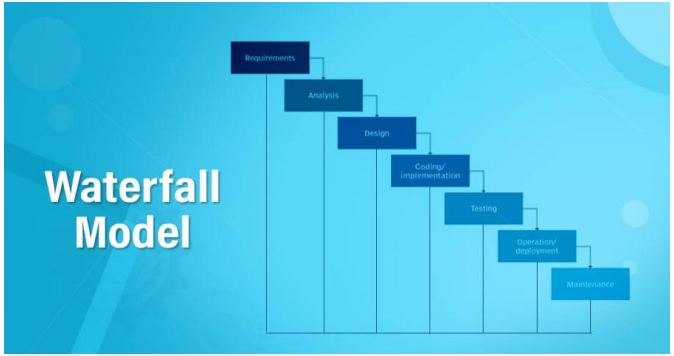
Plan-driven Model

- **Plan-driven model** is a more **formal specific approach** to creating an application.
- Plan-driven model considers the following aspects:
  - Consider defined plan, workflow, roles, responsibilities and work product descriptions, etc.
  - Process monitoring.
  - Focus on predictability.
  - Work through documentation.
  - Focus on verification and validation.
  - On-going risk management.
  - Defined up-front system architecture.
  - Extensive documentation.

Source: planio

Waterfall Model

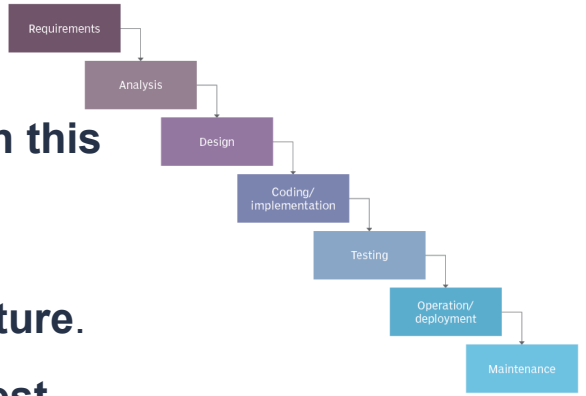
- **The Waterfall model** was the **first process model** introduced in 1960s.
- It is also known as a **linear-sequential life cycle model**.
- The Waterfall model has generally the following sequential phases: **Requirements**, **Analysis**, **Design**, **Implementation**, **Testing**, **Deployment** and **Maintenance**.
- Each phase **must be completed** before the next phase can begin and there is **no overlapping** in the phase.
- The Waterfall model allows to **manage every step of the development process** by a lot of **planning** and **documentation**.
- An end-user **isn't able to see a working product** until the late stages of development.



Source: eduCBA

Waterfall Model Phases

- **Requirement: All possible requirements** of the system to be developed are **captured in this phase**.
- **Analysis: The requirements** from the first phase are **studied in this phase** and Also **the design is prepared**.
- **Design**: The design helps in **specifying hardware and requirements** and also in defining the **overall system architecture**.
- **Implementation**: The implementation generally **takes the longest period of time** as it involved the actual development of the product.
- **Testing:** This phase **finds bugs and errors**. **Each unit is tested** for its functionality, which is referred to as **Unit Testing**.
- **Deployment**: The product is **deployed in the customer environment** or **released into the market**.
- **Maintenance**: It **fixes any issue after development** and also observes how the market reacts to the project.



Source: eduCBA

Pros and Cons of Waterfall Model

Pros

- The model can make to **meet deadlines** easily because specific instructions are already provided.
- It identifies deliverable and milestones, so **works well on mature deliverables**.
- The model is **easy to manage** as there is any external resources involved.
- It is best **suited for bigger teams and projects** due to rich documentation involved.
- It quickly **grasp the idea behind the project** and start producing quality code.
- It allows for **accurate cost estimation**.

Cons

- The model requires **heavy workload** during requirement **analysis** and **design**.
- **Changes** can be **extremely costly** or even impossible to make because it is very **difficult to go back** to any previous stage.
- It is **difficult to measure the progress** as there is **no working software** delivered until late in the development cycle.
- In a case customers don't know what they want, it may sometimes **lead to create a product that nobody desires**.

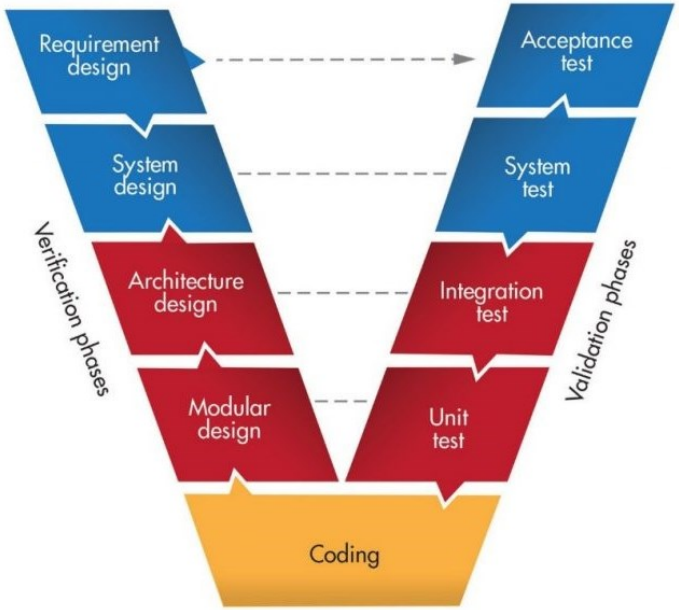
Systems with Waterfall Model

- **Embedded systems** where the software has to interface with hardware system.
  - It is not usually possible to delay decisions on the software's functionality due to the **inflexibility of hardware**.
- **Critical systems** where there is a need for extensive safety analysis of the software specification and design.
  - **The specification and design documents must be complete** so that this analysis is possible.
  - **Safety-related problems** in the specification and design are usually **expensive to correct a the implementation stage**.
- **Large software systems** that are part of broader engineering system **developed by several partner companies**.
  - Companies find it easier to use **rich documentations** for their development.

Source: eduCBA

V Model

- **The V model** is basically an extension of the Waterfall model, known as **Verification and Validation model**.
- Execution of processes in V model happens in a **sequential manner in a V-shape**.
- **The left side of the V** represents **verification** which involves in several **analysis and design** phases.
- **The right side of the V** represents **validation** which involves in several **testing** phases.
- The coding phase is **neutral** as it is **involved in both**.
- The V model demonstrates **the relationships** between **each phase of the development processes** and **its associated phase of testing**.
- The V model ensures that the results are **complete** and have **the desired quality**.



Source: eduCBA

Spiral Model Phases

- **Requirement design** (Acceptance tests are designed)
  - Requirement design as the first step in the verification processes **collects and analyses the requirements of the system**, and **establishes what the ideal system** hast to be perform.
- **System design** (System tests are designed)
  - System design is the phase where system engineers **analyse and understand the business of the proposed system** by studying the user requirements document and then **generate the software specification** which gives an **overall picture of how the system will looks like** and what it is going to do**.**
- **Architecture design** (Integration tests are developed)
  - Architecture design, as referred to **high-level design**, **maps out the technical approaches to build the system**, including the module based design of **computer architecture** and **software architecture**, such as interface relationships, database tables, architecture diagrams, etc.
- **Module design** (Unit tests are created)
  - Module design, as referred to **low-level design**, **include detail specifications** for how all functional and business logic will be implemented such as models, component, interfaces, and so forth.

Pros and Cons of V Model

Pros

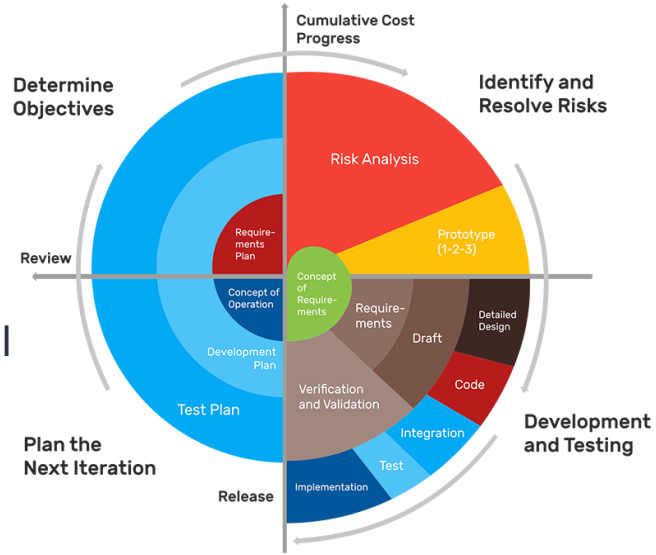
- This model is very **easy to understand and apply**.
- **Test activities** like planning are **clearly defined at early stage** before coding.
- The model **has higher chance of success** over the Waterfall model.
- It improve the system **quality throughout the validation** phases.
- The model can clearly **see what is ahead of project** and form precise **time and budget estimates**.
- It works well for bigger teams and projects due to **a clear image of system look like**.

Cons

- The model is **very rigid and little flexible** like the Waterfall model.
- It is **not suitable** for the project if there are **many changes required**.
- **No early prototype of the software** are produced until the implementation phase.
- **A lot of software development documentation** can be required.
- It is **not suitable for smaller projects** as it will take an **enormous amount of time to plan** the process and do the research.

Spiral Model

- **The Spiral model** is a **risk-driven** software development process model.
- It is **a combination of the Iterative model** with the systematic, controlled aspects of **the Waterfall model**.
- The Spiral model has four phases: **Planning**, **Risk Analysis**, **Construct**, **Evaluation**. Each phase of the Spiral model begins with a design goal and ends with the client reviewing the progress.
- The whole system build **goes through a couple of spiral** until the final system is developed. But, **the number of loops of the spiral is varied** depending on projects.
- If any risk is found during the risk analysis, alternative solutions are suggested by possibly **creating a prototype**.
- The Spiral model is used when software requires **often changes**, **continuous risk checks**, and **frequent releases**.



Source: Codegiant

Pros and Cons of Spiral Model

Pros

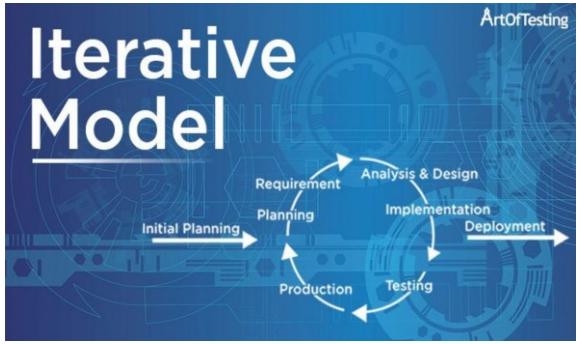
- The Spiral model **enhances avoidance of risk** due to high amount of risk analysis.
- Because of the extensive **use of prototype building,** cost estimation becomes easy.
- The Spiral model is **useful for large, complex and mission-critical projects**.
- It allows **early and frequent feedback** from users.
- **Additional functionality or changes** can be done at the relatively late stages.

Cons

- The Spiral model requires **more management** because process is complex.
- It can **be a costly model to use**.
- **Spiral** may go on **indefinitely** as end of project may not be known.
- It is **not suitable for small or low risk projects**.
- Risk analysis may **require highly specific expertise** as project's success is highly dependent on the risk analysis phase.

Iterative Model

- **The Iterative model enhances the evolving version iteratively** until the complete system is implemented and ready to be deployed.
- It is necessary to **break down** the entire project into **smaller chunks** for each iteration cycle.
- It typically **starts with a simple implementation** of a small set of the software requirements.
- **Each iteration cycle** goes through **the typical stages of the software development processes**: Requirements, Design, Implementation, Test and Deployment.
- The Iterative model allows to **access previous development cycles** which can be reviewed and changed.
- It may be possible to **work on a couple of iterations at once** if development resource is allowed.



Source: ArtOfTesting

Pros and Cons of Iterative Model

Pros

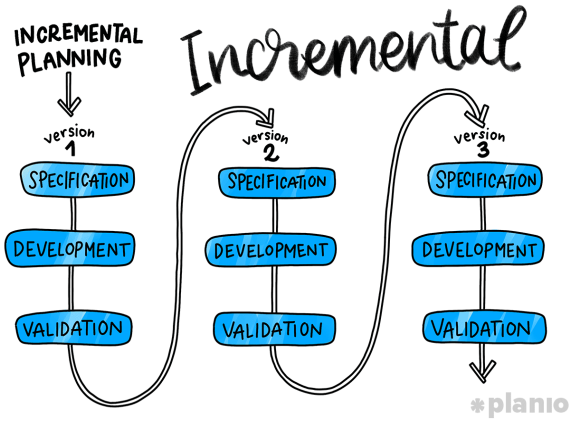
- The Iterative model **can make slight change** to the software.
- The Iterative model can allow **the parallel development**.
- **Some working functionality can be quickly developed** at early development.
- It is **less costly to change** the scope and requirements.
- **Testing** and **debugging** is **easy** during smaller iteration.
- It is **easier to manage risk** as high risk tasks are completed first.

Cons

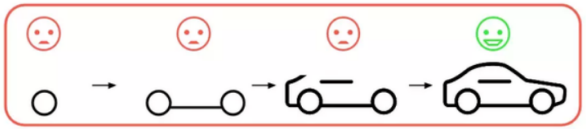
- The Iterative model **requires more resources** than the Waterfall model.
- It requires **overall more management attention** as constant management is required.
- It is **difficult to see a clear image** of how the end system should look like because of the iteration cycles.
- It is **not suitable for smaller projects** because the processes are difficult to manage.

Incremental Model

- **The Incremental model** aims to develop a software which **basic features in the first step** and incrementally developing the software or adding **new features in the further steps**.
- Each increment is **treated as a sub project** and follows **all the phases of the SDLC.**
- Once the increment is being developed, **the specific increment gets frozen** until the next increment.
- The Incremental model mostly used:
  - When some requirements are clearly specified but some requirements needs time.
  - When a new technology is used or required skillset is not available
  - When software engineering team is not highly skilled.
  - When a demand for an early release of the software is necessary.
  - When projects have lengthy development schedule.



Source: Planio



Pros and Cons of Incremental Model

Pros

- The Incremental model is **less costly** compared to others.
- The software will be **generated quickly** during SDLC.
- It is flexible and **less expensive** to change requirements and scope.
- **A customer can respond** to each incremental building.
- It is **easy to identify errors**.
- It is **easy to manage risk**.
- The Incremental model can **reduce delivery cost**.

Cons

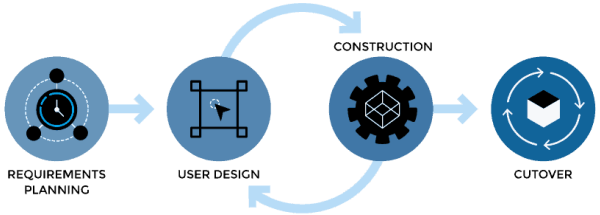
- The Incremental model **requires a good planning designing**.
- Each iteration phase is **rigid** and does not **overlap each other**.
- Modules for development should be **clearly defined**.
- It needs a **complete and clear understanding** of the whole software before it is broken down and build incrementally.
- **Total cost is much higher** than the Waterfall model.

Process Models and Agile Development

Agile Development

Rapid Development and Delivery

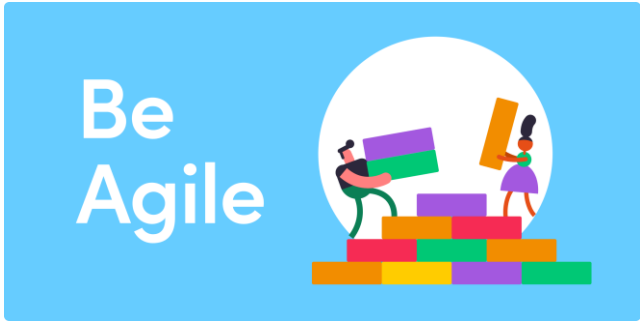
- **Rapid development and delivery** is now often the most important requirement for software systems.
  - Business operate in a **rapidly changing environment**.
  - It is practically **impossible to produce** a set of stable **software requirements**.
  - **Software has to evolve** quickly to **reflect changing business needs**.
- **Plan-driven development** is essential for some types of system, such as embedded systems, safety-critical control systems, but **does not meet these business needs**.
- Plan-driven development **completely specify the requirements** and then design, build and test a system.
- A **conventional waterfall** or **specification based process** is usually a **lengthy one**, and **the final software** is delivered to the customer **long after it was originally specified**.



Source: Plutora

Agile Development

- **Dissatisfaction with the overheads** such as extensive documentations and requirement analysis time involved in software design methods of the 1980s and 1990s **led to the creation of agile development**.
- Agile development methods emerged in the late 1990s whose aim was to **radically reduce the delivery time** for working software systems.
- The agile development:
  - **Focus on the software itself** rather than the design and documentation.
  - Are **based on an iterative approach** to software development
  - Are intended to **deliver working software quickly** and **evolve this quickly** to meet changing requirements.



Source: monday.com

Agile Development

- The aim of agile development is to **reduce overheads in the software process** and to **be able to respond quickly to changing requirements** without excessive rework.
- Unlike Waterfall, the processes of specification, design and implementation are **inter-leaved**.
- The system using agile development is developed as **a series of increments with stakeholders** involved in specifying specification and evaluation.
- Agile development has **frequent delivery of new versions** for evolution.
- **Extensive tool support** (e.g. automated testing tools, tools to support configuration management) is used to support the development process in agile development.
- It **minimises documentation** by using **informal communications** to focus on working code.

Source: eduCBA

Agile Development Manifesto

- The philosophy behind agile development is reflected in the **agile manifesto** (http://agilemanifesto.org) issued by the leading developers of these methods.
- The manifesto states **four values**;

*"We are **uncovering better ways to developing software** by doing it and helping others do it. Through this work we have come to value:*

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan.

That is, while there is value in the items on the right, we value the items on the left more."

Source: Lynne Cazaly

Agile Development Principles

- The **Manifesto for Agile development** is based on **12 principles**.
  - **Customer satisfaction** by early and **continuous delivery** of valuable software.
  - **Welcome changing requirements**, even in late development.
  - **Deliver working software** frequently (weeks rather than months)
  - **Close, daily cooperation** between business people and developers.
  - Projects are built around **motivated individuals, who should be trusted**.
  - **Face-to-face conversation** is the best form of communication (co-location).
  - **Working software** is the **primary measure** of progress.
  - **Sustainable development**, able to maintain a constant pace.
  - **Continuous attention** to technical excellence and good design.
  - **Simplicity** – the art of maximizing the amount of work not done – is essential.
  - Best architectures, requirements, and designs emerge from **self-organizing teams**.
  - **Regularly, the team reflects** on how to become more effective, and adjusts accordingly.

Agile Development Principles in Common

Principle	Description
Customer Involvement	**Customers should be closely involved throughout the development process**. Their role is to provide and prioritise new system requirements and to evaluate the iterations of the system.
Incremental Delivery	**The software is developed in increments** with the customer specifying the requirements to be included in each increment.
People not Process	**The skills of the development team should be recognised and exploited**. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace Change	**It is expected that the system requirements to change** and so design the system to accommodate these changes.
Maintain Simplicity	**It should focus on simplicity in both the software being developed and in the development process**. Wherever possible, actively work to eliminate complexity from the system

Agile Development Applicability

- Agile development have been **particularly successful for two kind of system development**.
  - **Product development** where a software company is developing **a small or medium-sized product for sale**. Generally, most of software products and apps are now developed using an agile approach.
  - **Custom system development**,
    - ✓ where there is **clear commitment from the customer** to become involved in the development process.
    - ✓ where there are **less external rules and regulations** that affect the software.
- Agile development also **works well in the situations** which **continuous communication is required** between the product manager or system customer and the development team.

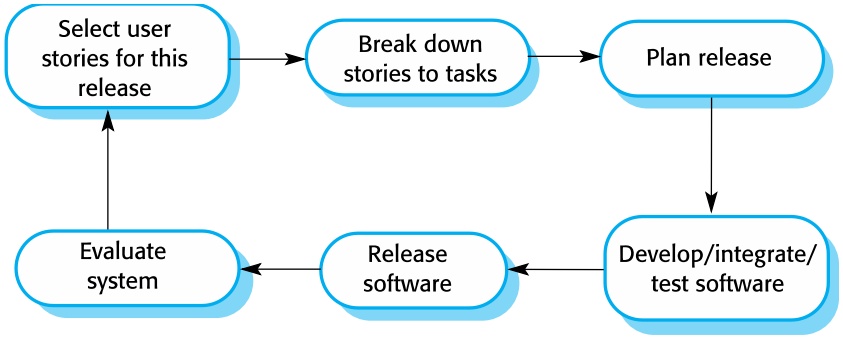
Source: Bydrec Blog

Process Models and Agile Development

Extrme Programming
(Agile Development Techniques)

Extreme Programming (XP)

- The ideas underlying agile methods were developed in the late 1990s and introduced a range of agile development techniques.
- **The most significant approach** to changing software development culture was the development of **Extreme Programming (XP)**.
- Extreme Programming (XP) takes an **'extreme' approach to iterative development** such as
  - **Several new versions of a system may be developed** by different programmers, integrated, and tested in a day.
  - **Programmers work in pairs** and **develop tests for each task** before writing the code.
  - **All tests must be run for every build** and the build is **only accepted if tests run successfully**.



Extreme Programming Practices (1)

Principle or Practice	Description
Incremental Planning	By creating an essential foundation to start with, **the development team can incrementally add to change and evolve the development** by reflecting on the strengths and weakness of the development.
Small Releases	**The minimal useful set of functionality** that provides business value **is developed first**. Releases of the system are frequent.
Simple Design	**Enough design** is carried out to **meet the current requirements and no more**.
Test-first Development	**An automated 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 (2)

Principle or Practice	Description
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 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 it often causes 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 brining system requirements to the team for implementation.

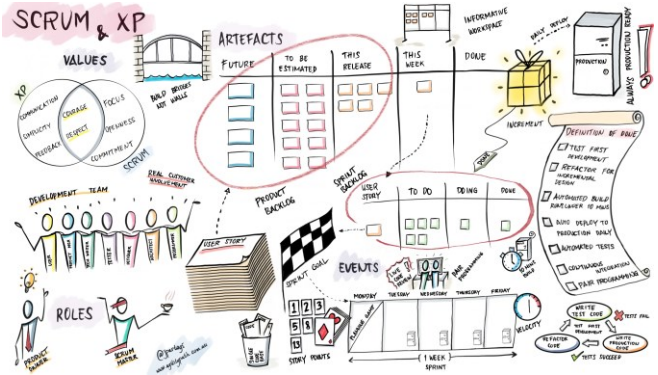
XP and Agile Principles

- The practices from XP reflect the principle of the agile manifesto:
  - **Incremental development** is supported through **small, frequent system releases**.
  - **Customer involvement** is supported through the **continuous engagement of the customer** and the development team.
  - **People, not process**, are supported through **pair programming**, **collective ownership of the system code**, and **sustainable development process** that avoids long working hours.
  - **Change** is embraced through **regular system releases** to customers, **test-first development**, **refactoring** to avoid code degeneration, **continuous integration** of new functionality.
  - **Maintaining simplicity** is supported by **constant refactoring** that improves code quality and by **using simple designs**.

Source: AltexSoft

Influential XP Practices

- Extreme programming has a **technical focus** and **is not easy to integrate with management practice** in most organisations.
- Companies adopting agile development **pick and choose some XP practices** that are most appropriate for their way of working.
- More commonly, the XP practices are **used in conjunction with a management-focused agile method** such as Scrum.
- Some important key practices are:
  - **User stories** for specification
  - **Refactoring**
  - **Test-first development**
  - **Pair programming**



Source: AltexSoft

XP Practices – User Stories for Requirements

- 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 **user stories** or **scenarios**.
- User stories or scenarios 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 implementation** in the next release **based on their priorities and the schedule estimates**.
- So, the intention is to **identify useful functionality ASAP** and the next release of the system will have the functionality.
- However, **the principal problem with user stories is completeness**.
  - It is **difficult to judge if enough user stories have been developed** to cover all of the essential requirements of a system.
  - It is **difficult to judge if a single story give a true picture of a activity**.

XP Practices – User Stories with Mentcare System

- There is a story card for the **Mentcare** system, and also are tasks from the story.

Prescribing medication

The record of the patient must be open for input. Click on the medication field and select either 'current medication', 'new medication' or 'formulary'.

If you select 'current medication', you will be asked to check the dose; If you wish to change the dose, enter the new dose then confirm the prescription.

If you choose, 'new medication', the system assumes that you know which medication you wish to prescribe. Type the first few letters of the drug name. You will then see a list of possible drugs starting with these letters. Choose the required medication. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

If you choose 'formulary', you will be presented with a search box for the approved formulary. Search for the drug required then select it. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

In all cases, the system will check that the dose is within the approved range and will ask you to change it if it is outside the range of recommended doses.

After you have confirmed the prescription, it will be displayed for checking. Either click 'OK' or 'Change'. If you click 'OK', your prescription will be recorded on the audit database. If you click 'Change', you reenter the 'Prescribing medication' process.

Task 1: Change dose of prescribed drug

Task 2: Formulary selection

Task 3: Dose checking

Dose checking is a safety precaution to check that the doctor has not prescribed a dangerously small or large dose.
Using the formulary id for the generic drug name, lookup the formulary and retrieve the recommended maximum and minimum dose.
Check the prescribed dose against the minimum and maximum. If outside the range, issue an error message saying that the dose is too high or too low.
If within the range, enable the 'Confirm' button.

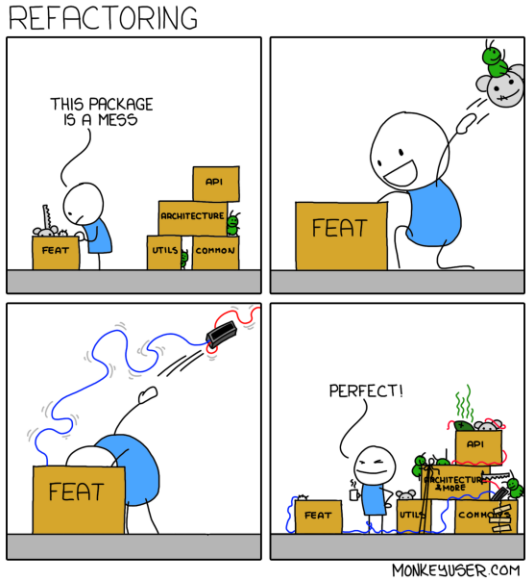
XP Practices – Refactoring (1)

- One of principles in conventional software engineering **considers design for change**.
- It is typically **worth spending time and effort anticipating future changes** to software design as **this reduces costs later** in the life cycle.
- However, **XP considers that designing for change is not worthwhile** to add generality to a program code because **changes will always have** to be made to the code being developed.
- Rather, the developers of XP proposes **constant code improvement (refactoring) to make change easier** when they have to be implemented.
- Thus, refactoring in XP means that **the programming team looks for possible improvement** to the software and **implements them immediately**.

Source: Datree

XP Practices – Refactoring (2)

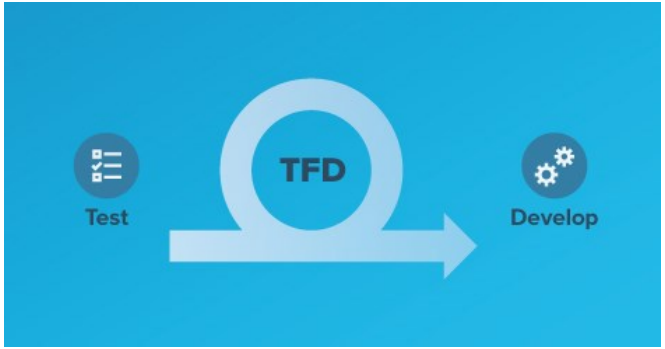
- A fundamental problem of incremental development is that **local changes tend to degrade the software structure**.
- Consequently, further changes to the software **become harder to implement and code is often duplicated**.
- Refactoring **improves the software structure and readability** and so avoid the structural drop that naturally occurs when software is changed.
- Examples of refactoring include:
  - The **reorganisation of a class hierarchy** to remove duplicate code.
  - The tidying up and **renaming of attributes and methods**.
  - The **replacement of similar code sections**, with calls to methods defined int a program library.
- In practice, sometimes, **development pressure delays refactoring** due to the implementation of new functionality.



Source: Pinterest

XP Practices – Test-first Development (1)

- **In incremental development**, there is **no overall system specification** that can be used by an external testing team to develop **system tests**.
- Typically, some approaches to incremental development have a very **informal testing process**.
- In XP, **testing is automated** and development **cannot proceed** until all tests have been **successfully executed**.
- The key features of testing in XP are:
  - **Test-first development**
  - **Incremental test development** from scenarios
  - **User involvement** in test development and validation
  - The use of **automated testing frameworks**



Source: QMetry

XP Practices – Test-first Development (2)

- XP's test-first philosophy has now **evolved into more general test-driven development** techniques.
- **Write the tests before writing the code**, instead of writing code and then writing tests for that code.
- **Tests are written as programs** rather than data so that they can **be executed automatically**.
- In XP, the tasks are the principal unit of implementation and so **each task generates one or more unit tests** that check the implementation.
- When new functionality is added, **all previous and new tests are run automatically**.
- The example is a shortened description of a test case to check that the prescribed dose of a drug does not fall outside known safe limits.

Test 4: Dose checking

Input:
1. A number in mg representing a single dose of the drug.
2. A number representing the number of single doses per day.

Tests:
1. Test for inputs where the single dose is correct but the frequency is too high.
2. Test for inputs where the single dose is too high and too low.
3. Test for inputs where the single dose * frequency is too high and too low.
4. Test for inputs where single dose * frequency is in the permitted range.

Output:
OK or error message indicating that the dose is outside the safe range.

XP Practices – Test-first Development Customer Involvement

- **The role of the customer** in the testing process is to **help developing acceptance tests** for the user stories that are to be implemented in the next release of the system.
  - **Acceptance test** is the process whereby the system is tested using customer data to check that **it meet's customer's needs**.
- **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.

Source: Altervista

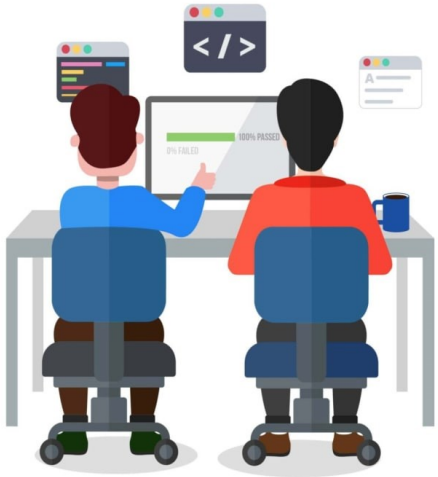
XP Practices – Test-first Development Problems

- Test-first development and automated testing usually result in **a large number of testing being written and executed**.
- However, there are problems in test-first development:
  - **Programmers prefer programming to testing** and sometimes they take shortcuts when writing test. For example, they may write incomplete tests that do not check for all possible exceptions that may occur.
  - **Some tests can be very difficult to write incrementally**. For example, in a complex user interface, it is often difficult to write unit tests for the code that implements the '**display logic**' and **workflow between screens**.
- It is **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.
- It **might be evitable to find undetected bugs** after the system release.

Source: LinkedIn

XP Practices – Pair Programming (1)

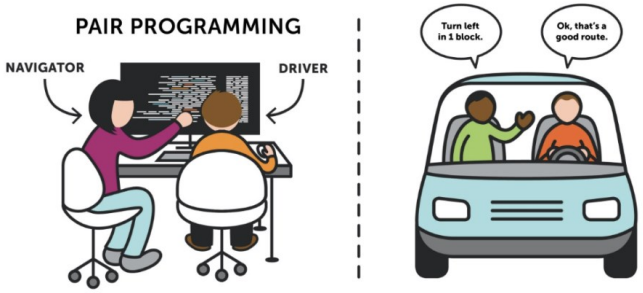
- Pair programming involves **programmers working in pairs**, **developing code together**.
- The same pair do not always program together but **pairs are created dynamically**. So, all team members work with each other during the development process.
- Pair programming has a number of advantages:
  - It supports the idea of **collective ownership and responsibility** for the software. The team owns the software and individuals are not held responsible for problems with the code.
  - It serves as **an informal review process** as each line of code is looked at by at lease two people. It is cheaper and easier to organise than formal program inspections.
  - It **encourages refactoring** to improve the software structure. So, the whole team can benefit from the refactoring.



Source: Fiverr

XP Practices – Pair Programming (2)

- However, many companies that have adopted agile methods are **suspicious of pair programming** and do not use it as they think **the loss of human resources**.
- Other companies mix **pair and individual programming** with an experienced programmer working with a less experienced colleague.
- Actually, pair programming **is not necessarily inefficient** and there is some evidence suggesting that a pair working together is **more efficient than 2 programmers working separately**.
  - The reasons suggested are that pairs discuss the software before development and so probably **have fewer false starts and less rework**.
  - **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.



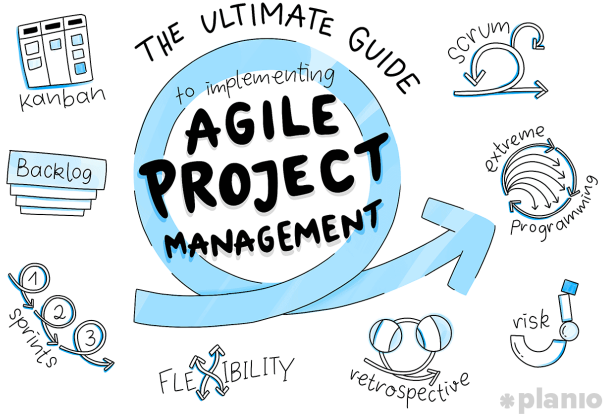
Source: Unruly

Process Models and Agile Development

Scrum
(Agile Project Management)

Agile Project Management

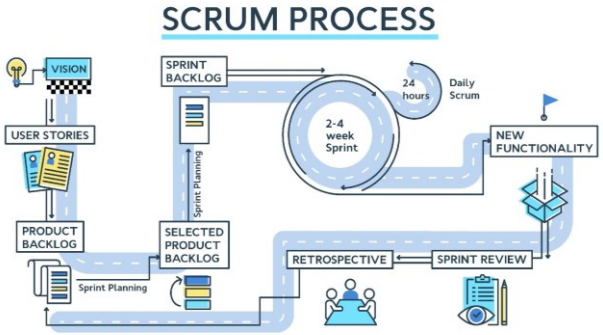
- The principal **responsibility of software project managers** is to manage the project so that the software is **delivered on time and within the planned budget** for the project.
- Managers draw up a plan for the project showing:
  - **What** should be delivered
  - **When** it should be delivered
  - **Who** will work on the development of the project deliverables.
- **Plan-driven approaches** to software development **are used to meet managers' need** as there is requirement and design clearly defined.
- Unlike plan-driven development, agile project management requires a different approach, which is **adapted to incremental development** and **iterative development**.



Source: TriQiQ

Scrum

- **Scrum** is an **agile development method to provide a framework** for organising agile projects.
- Scrum follows the principles from the **agile manifesto**.
- It focuses on **managing iterative development** rather than **specific agile practices** such as pair programming and test-first development.
- There are three phases in Scrum:
  - **The initial phase is an outline planning phase** where you establish the **general objectives** for the project and design the **software architecture**.
  - This is followed by **a series of sprint cycles**, where each cycle develops **an increment of the system**.
  - **The project closure phase** wraps up the project, completes required documentation such as user manuals, and accesses the lessons learned from the project.



Source: Crystalloids

Scrum Terminology (1)

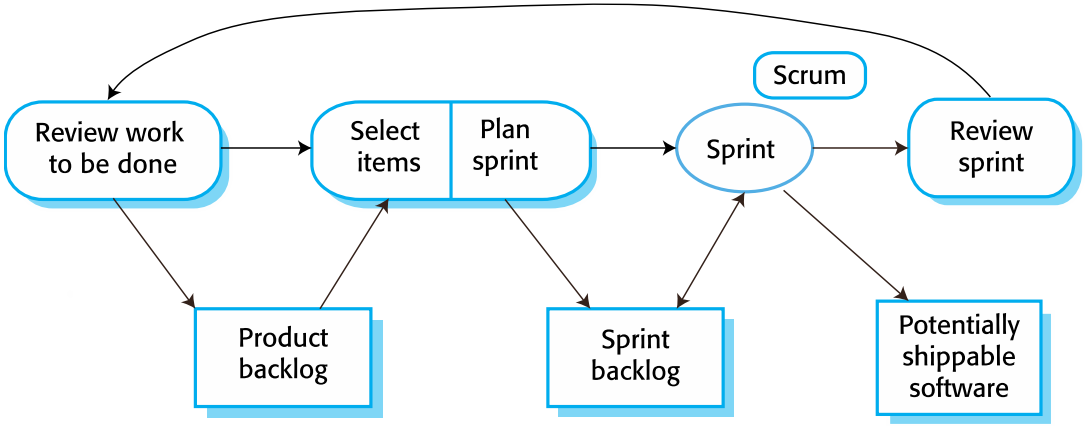
Scrum Term	Definition
Development Team	**A self-organising group of software developers**, which should be **no more than 7 people**. They are responsible for developing the software and other essential project documents.
Potentially Shippable Product Increment	**The software increment is delivered from a sprint**. The idea is that this should be 'potentially shippable', meaning that it is a **finished state** and no further work such as testing is needed to incorporate it into the final product
Product Backlog	**This is a list of 'to do' items which the Scrum team must tackle**. They may be feature definitions for the software requirements, user stories or descriptions of supplementary tasks that are needed, such as architecture definition or user documentation
Product Owner	An individual (or possibly a small group) whose job is to **identify product features or requirements**, **prioritise these for development** and continuously **review the product backlog** to ensure that the project continues to meet critical business needs. **The Project Owner can be a customer but might also be a product manager in a software company or other stakeholder representative**.

Scrum Terminology (2)

Scrum Term	Definition
Daily Scrum	**A daily meeting of the Scrum team that reviews progress and prioritises work to be done that day**. Ideally, this should be a short **face-to-face meeting** that includes the whole team. The meeting should be less than 30 minutes.
Scrum Master	**The Scrum master is a responsible** for ensuring that **the Scrum process is followed and guides the team in the effective use of Scrum**. The Scrum master is responsible for **interfacing with the rest of the company** and for **ensuring that the Scrum team is not diverted by outside interference**. The Scrum master should not be thought of as a project manager.
Sprint	**A development iteration. Sprints are usually 2 to 4 weeks long**.
Velocity	**An estimate of how much product backlog effort that a team can cover in a single sprint**. Understanding a team's velocity helps them estimate **what can be covered in a sprint** and provides a **basis for measuring improving performance**.

Scrum Sprint Cycle

- The typical Scrum process or sprint cycle is shown here.
- Sprints are **fixed length of time**, normally between **2 and 4 weeks**.
- **The starting point for planning is the product backlog**, which is the list of work to be done on the project.
- The product backlog includes **product features**, **requirements**, and **engineering improvement** that have to be worked on by the Scrum team.
- Each sprint cycle **produces a product increment** that could be delivered to customers.



Scrum – Teamwork

- The **Scrum Master** is a facilitator who **arrange daily meetings**, **tracks the backlog** of work to be done, **records decisions**, **measures progress** against the backlog, **communicates with customers**, etc.
- The whole development team **attends short daily meetings (Daily Scrums)** 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 can re-plan short-term work to scope with them if problems arise.
- There is a **review meeting** at the end of each sprint.
  - The team reviews **the way they have worked** and reflects on **how thing could have been done better**.
  - It provides **input for the product backlog** for the next sprint.

Source: SolDevelo

Scrum – Benefits

- The product is **broken down into a set of manageable and understandable chunks** that stakeholder can relate to.
- Unstable requirements do not hold up progress so it **focuses on doing tasks defined clearly** in order to not waste time and cost.
- **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. They are not faced with late-minute surprise when a team announces that software will no be delivered as expected.
- **Trust between customers and developers** is established, and **a positive culture is created** in which everyone expects the project to succeed.

Larger projects are broken down into **more manageable chunks.**

The scrum approach ensures **efficient use of client time and money.**

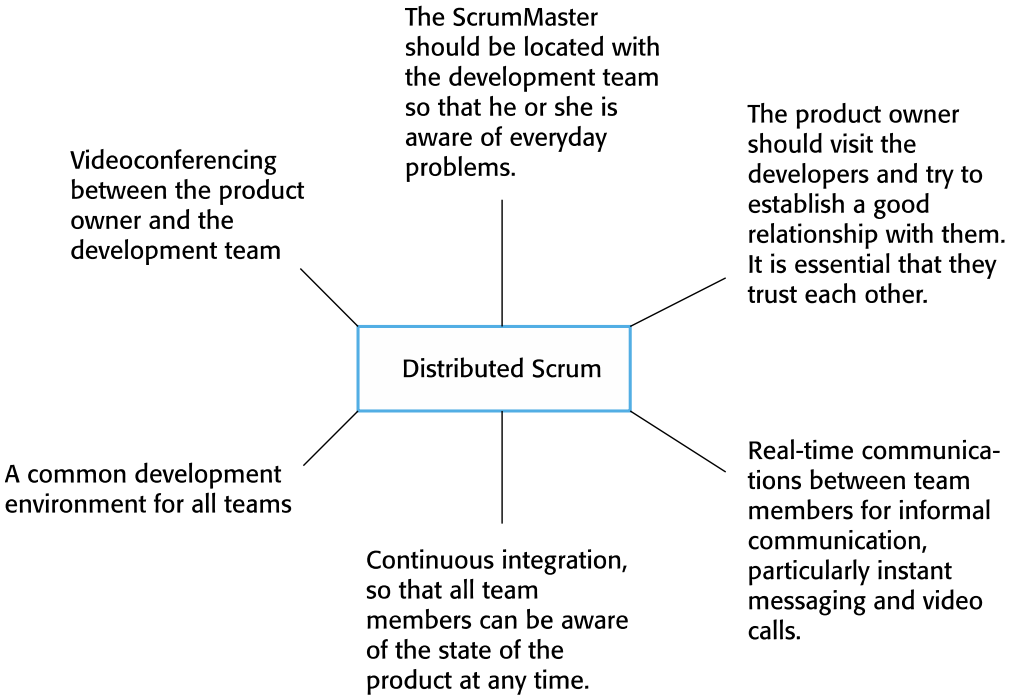
The development team and the client work in **closer contact** and **receive more feedback.**

The process allows for **increased flexibility.**

Source: fraxent

Distributed Scrum

- **Scrum**, as originally designed, **was intended for use with co-located teams** where all team members could get together every day in stand-up meetings.
- However, much software development now involved **distributed team**, with team members located **in different places around the world**.
- This allows companies to take advantage of **lower cost staff**, access to **specialist skills** possible, and allow for **24-hour development**.
- The example shows the requirements for Distributed Scrum.



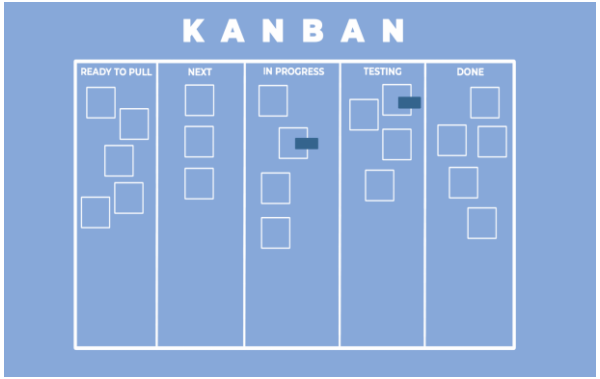
Process Models and Agile Development

Kanban and RAD

WMG
THE UNIVERSITY OF WARWICK

Kanban Model

- **Kanban (**meaning signboard) suggests an approach of **managing the flow of work** with an emphasis on **continuous improvement**.
- Kanban was initially invented as a way of **managing just in time manufacturing processes**.
- It is designed to **optimize workflow** and **use team's full capacity**.
- There are **4 Kanban principles**: 1) Start with what you do now, 2) Agree to pursue incremental, evolutionary change, 3) Respect the current process, roles, responsibilities & titles, 4) Encourage acts of leadership all levels in your organisation.
- There are **6 Kanban practices**: 1) Visual the workflow, 2) Limit work in progress, 3) Manage flow, 4) Make process policies explicit, 5) Feedback loops, 6) Improve collaboratively.



Source: Infinity

Pros and Cons of Kanban Model

Pros

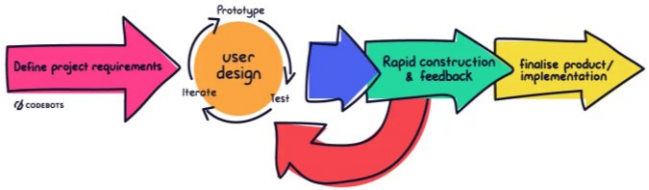
- Kanban is **easy to detect bottlenecks** in development workflow.
- Kanban **assists just-in-time approach** to improve the flow and management inventory.
- Kanban **supports visualisations of the process outputs** which makes the review of work easier.
- Kanban **enhances the effectiveness of human resource** a company.
- Kanban **supports continuous and sustainable improvements** in production of the company.

Cons

- Kanban will **become very difficult** to apply **if too much activities or tasks** are interrelated in a system.
- **The prediction of specific timelines** for completion of tasks **become difficult** as the tasks are continuously shifted.
- It is **less effective in shared-resource situations** like each activity may require the same resource.
- Kanban with **outdated activities** can make the board **overcomplicated**.

RAD (Rapid Application Development) Model

- **The RAD Model** is very **similar to the Iterative model** which consists of iterations, known as prototypes**.**
- It is based on **prototyping without much specific planning**.
- It allows to **deploy each prototype separately** and thus **collect feedback** throughout development.
- **Every prototype** is developed in a **reusable way** for future projects. So, this saves time and money.
- The RAD embraces change and enables to **adapt to the changing environment**.
- The RAD distributes the **analysis**, **design**, **build** and **test** phases into a series of short iterative development cycles.



Source: Codebots

Pros and Cons of RAD Methodology

Pros

- The RAD model **enhance flexibility and adaptability** as developers can make adjustments quickly.
- Quick iterations **reduce development time** and **speed up delivery**.
- Better risk management as stakeholders can **discuss and address code vulnerabilities**.
- It **increases reusability** of components meaning **less manual coding**, **less room for errors**, and **shorter testing times**.
- The RAD model **includes integrations early** on in the software development process.

Cons

- The RAD model **requires highly skilled** developers/designers.
- **Management complexity** is more.
- It is **suitable for** project requiring **shorter development times**.
- If the software **can't broken down** into smaller chunks, it will be **hard time utilising the SDLC**.
- It is also **required the client to be constantly involved** in the development stage.

Process Models and Agile Development

Scaling Agile Development

Scaling Agile Development

- **Agile development** has proved to be **successful for small and medium sized projects** that can be **developed by a small co-located team**.
- **Scaling up agile development** involves to **cope with larger, longer projects** where there are **multiple development teams**, perhaps **working in different locations**.
- Scaling agile development has the following related aspects:
  - **Scaling up** agile development to **handle the development of large system** that are too big to be developed by a single small team.
  - **Scaling out** agile development from **specialised development to more widespread use in large company** that has many years of software development experience.
- When scaling agile development, it is important to **keep agile fundamentals** such as flexible planning, frequent system releases, continuous integration, test-driven development, etc.

Source: SolDevelo

Scaling Agile Development – Practical Problems Issues

- Agile development may **not be suitable** for **large and complex systems**.
- **Agile development for large and long-lifetime systems** may have the following problems:
  - Agile development is most appropriate for new software development rather than software maintenance. Yet, **the majority of software costs in large companies come from maintaining their existing software systems**.
  - Agile development is designed for small co-located teams, yet **much software development now involves worldwide distributed teams**.
  - **The informality of agile development is incompatible with the legal approach to contract definition** that is commonly used **in large companies**.

Source: Relevant Software

Scaling Agile Development – Contractual Issues

- **Contractual issues can be another major problem** when agile development is used.
- **Most software contracts** for custom systems are **based around a specification**, which sets out **what has to be implemented** by the system developer for the system customer.
- However, although interleaved development of requirements and code is fundamental to agile development, it is **difficult to include new requirements** in the contract later.
- When using agile development, **a contract that pays for developer time rather than functionality**:
  - As long as all goes well, this benefits both the customer and the developer.
  - However, if problems arise, there may be difficult disputes over who is to blame and who should pay for the extra time and resources.

Source: Brunswick Companies

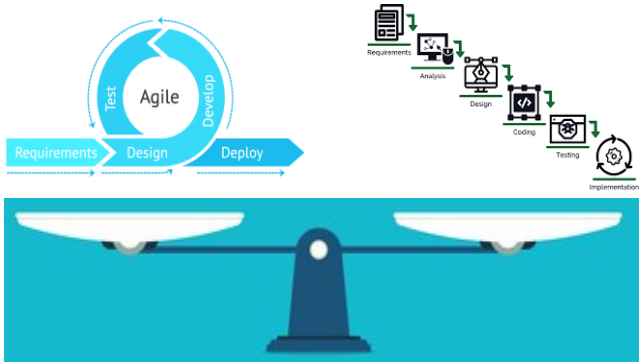
Scaling Agile Development – Software Maintenance Issues

- Most organisations **spend more on maintaining existing software** than they do on new software development.
- So, if agile development is to be successful, they **have to support maintenance** as well as original development.
- There are three type of problems in maintenance with agile development:
  - Agile development collects requirements informally and incrementally and **do not create coherent requirement document**.
  - While customers may justify the full-time involvement during the development, **this is less involvement during maintenance** because they are likely to loose interest in the system.
  - Agile development relies on team members understanding aspects of the system without having to consult documentation. For long-lifetime systems, this is a real problem as **the original developers will not always work on the system**.

Source: Brin Software Limited

Scaling Agile Development – Balances Issues

- Most large agile software development projects **combine practices from plan-driven and agile approaches**.
- Deciding on the balance between a plan-based and an agile development 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 development.
  - **Is an incremental delivery strategy, where you deliver the software to customers and get rapid feedback?** If so, consider using agile development.
  - **How large is the system that is being developed?** Agile development is 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 development may have to be used.



Scaling Agile Development – System Attributes Issues

- Some of systems have to **highly consider their attributes** such as **size**, **complexity**, **real-time response** and **external regulation**.
- These system importantly needs to be **some up-front planning, design, and documentation in the system engineering process**.
- Some of the key issues on systems are:
  - **How large is the system being developed?** Agile development is most effective a relatively small co-located team who can communicate informally. This may not be possible for large system that required larger developer teams. So, a plan-driven development may have to be used.
  - **What type of system is being developed?** Systems that require a lot of analysis before implementation need a fairly detailed design to carry out this analysis.
  - **What is the expected system lifetime?** Long-lifetime systems may require more documentation to communicate to communicate the original intentions of the system developers to the support team.
  - **Is the system subject to external regulation?** If a system has to be approved by an external regulator, you will probably be required to produce detailed documentation as part of the system safety case.

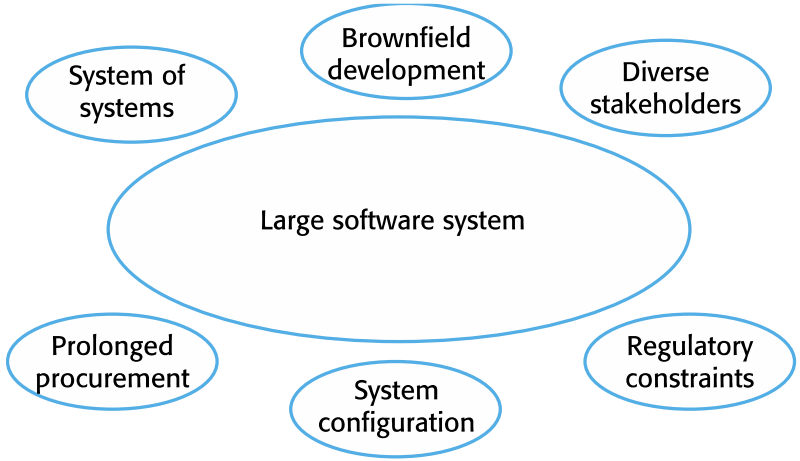
Scaling Agile Development – People and Teams Issues

- Agile development requires **the development team to cooperate and communicate** during the development.
- Agile development **relies on individual engineering skills** and **software support** for the development process.
- Some key issues on development team are:
  - **How good are the designers and programmers in the development team?** It is sometimes argued that agile development requires higher skill levels than plan-based development in which programmers simply translate a detailed design into code.
  - **How is the development team organised?** Design documents may be required to communicate if the development team is distributed or part of the development is being outsourced.
  - **What support technologies are available?** Agile development often relies on good tools to keep track of an evolving design. If IDE does not have good tools for program visualisation and analysis, more design documentation may be required.

Source: VectorStock

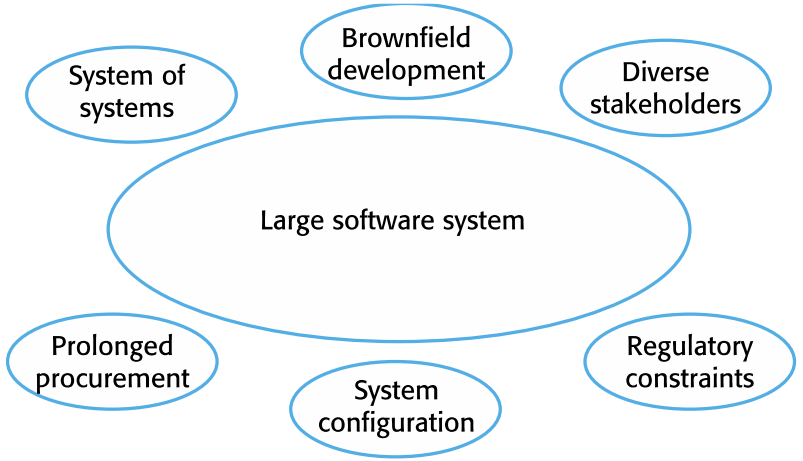
Scaling Agile Development – Large Systems Issues (1)

- **Large systems are usually system of systems** where separate teams develop each system. Frequently, these teams are **working in different places, sometimes in different time zones**.
- **Large systems are brownfield systems**, that is they 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.



Scaling Agile Development – Large Systems Issues (2)

- **Large systems and their development processes are often constrained by external rules and regulations** limiting the way that they can be developed, that require certain type of system documentation to be produced.
- **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 with different perspectives and objectives**. It is practically impossible to involve all of these different stakeholders in the development process.



Scaling Agile Development – Organisation Issues

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

Source: Association for Talent Development

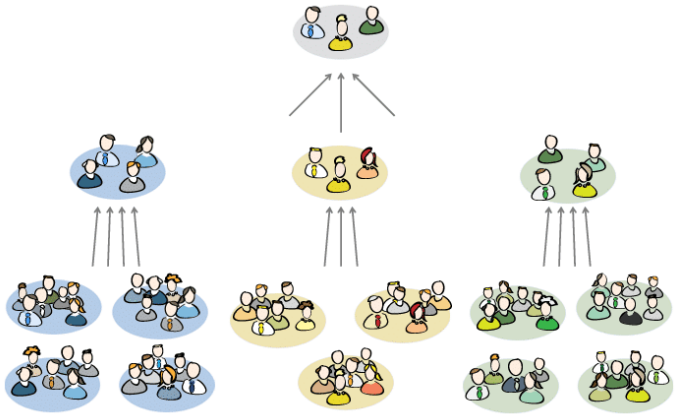
Scaling Up to Large Systems

- **No single model is appropriate for all large-scale agile product** because the type of product, the customer requirements, and the people available are all different.
- Approaches to scaling agile development have **a number of things in common**:
  - **A completely incremental approach** to requirements engineering is **impossible**.
  - There **cannot be a single product owner** or customer representative.
  - For large systems development, it is **impossible to focus only on the code** of the system.
  - **Cross-team communication mechanisms** have to be designed and used.
  - **Continuous integration** is practically **impossible**. However, it is essential to **maintain frequent system builds and regular release** of the system.

Source: Boston Consulting Group

Multi-team Scrum

- **Multi-team Scrum** has been adapted for **large-scale development**.
- The key characteristics of multi-team Scrum are:
  - **Role replication**: Each team has a Product Owner for their work component and Scrum master.
  - **Product architects**: Each team chooses a product architect and these architects collaborate to design and evolve the overall system architecture.
  - **Release alignment**: The dates of product releases from each team are aligned so that a demonstrable and complete system is produced.
  - **Scrum of Scrums**: There is a **daily Scrum of Scrums** where **representatives from each team** meet to **discuss progress and plan work** to be done. Individual team Scrums may be staggered in time so that representatives from other teams can attend if necessary.



Source: microTOOL

Summary

- There are typically two types of software development process models: **plan-driven models** and **agile models**. Some example in those models are **Waterfall Model**, **Rapid Prototyping Model**, **Spiral Model**, **Scrum Model**, **XP Model** etc.
- The Waterfall model has generally the following sequential phases: **Requirements**, **Analysis**, **Design**, **Implementation**, **Testing**, **Deployment** and **Maintenance**.
- Agile development are **iterative development** methods that focus on **reducing process overheads and documentation** and on **incremental software delivery**.
- The decision on whether to use an agile or a plan-driven 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.

Source: Shutterstock

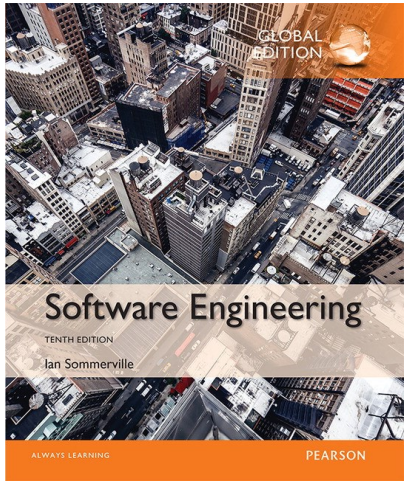
Summary

- Agile values include **Individuals and interactions** over processes and tools, **Working software** over comprehensive documentation**, Customer collaboration** over contract negotiation, **Responding to change** over following a plan.
- **Scrum** 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.
- Many practical development methods are a **mixture of plan-based and agile development**.
- To scale agile development, some plan-based practices have to be integrated with agile practices, including **Up-front requirements**, **Multiple customer representatives**, **More documentation**, **Common tooling** across project teams, **The alignment of release across teams**, etc.

Source: Shutterstock

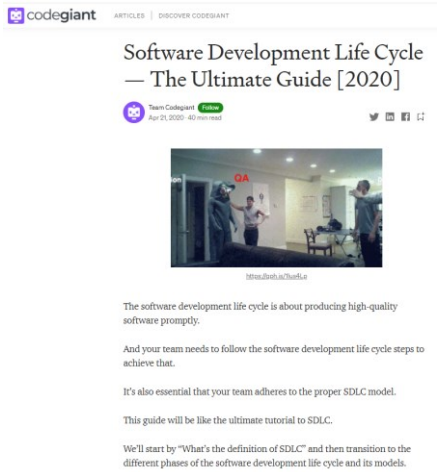
REFERENCE

[Main References]



Software Engineering
Ian Sommerville

Agile software development



codegiant
Team Codegiant

Software Development Life Cycle – The Ultimate Guide

[Other References]

- Wikipedia:
  - https://en.wikipedia.org/wiki/Waterfall_model
  - https://en.wikipedia.org/wiki/V-Model
  - https://en.wikipedia.org/wiki/Rapid_application_development
  - https://en.wikipedia.org/wiki/Agile_software_development
  - https://en.wikipedia.org/wiki/Scrum_(software_development)
  - https://en.wikipedia.org/wiki/Extreme_programming
  - https://en.wikipedia.org/wiki/Spiral_model
  - https://en.wikipedia.org/wiki/Iterative_and_incremental_development
- W3Schools: https://www.w3schools.in/sdlc-tutorial/v-model/
- TutorialsCampus: https://www.tutorialscampus.com/sdlc/incremental-model.htm
- Djaa: https://djaa.com/principles-and-general-practices-of-the-kanban-method/
- Accounting: https://www.accountingformanagement.org/kanban/
- Zentao: https://www.zentao.pm/agile-knowledge-share/Scrum-what-%23039%3Bs-the-difference-between-incremental-and-iterative-development-845.mhtml
- Agilemanifsto.org: http://agilemanifesto.org/

THANKS!

Any questions?

y.s.park@warwick.ac.uk