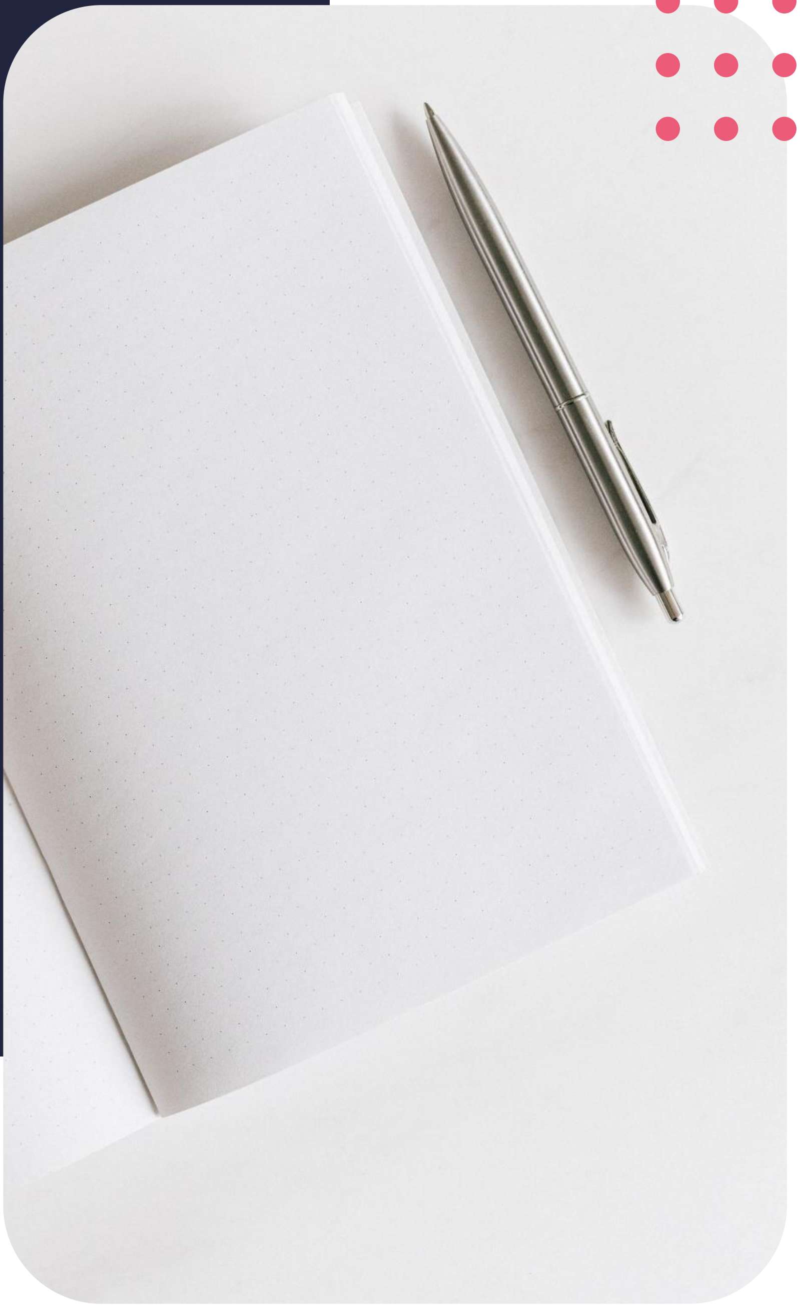
Professional Diploma in Computer Science

Agile Software Development Methods

Module 4 Lesson 3

Summary Notes

Contents



Lesson objectives

By the end of this lesson, you should be able to:

- Appreciate the shortcomings of traditional software design methodologies
- Examine how agile software development methodologies speed up software development
- Understand how different software development strategies under agile development are used
- Distinguish between different software development strategies

Introduction

Welcome to our third lesson in our final module. In our previous lesson, we looked at software development methodologies that were used from the inception of computer software. These have helped speed up the development process and resulted in far less errors than what was previously the case. Despite that lesson being quite long, we still did not do enough justice to the software development life cycle! In this lesson, we are going to carry on from where we left off, but in a slightly different direction, as we are going to be looking at methods that are being used in the modern world. Software has pretty much moved away from traditional development methods, and as such, it brought the need for new methods to be employed. We are going to be taking a closer look at each of these methods, with special emphasis on the most popular ones.

Agile development methods

If you remember from our last lesson, one of the biggest pitfalls of the almighty waterfall model was its rigidity. It proved very costly to make changes to aspects of the project once it had kicked off. The agile methodology attempts to fix this by anticipating change and the software development process, which allows for much more flexibility than what traditional methods could offer. Agile methods are based on giving first priority to customer participation and the project, right from the get-go, up until we have a finished, working product.

Life before agile

Agile software development really sounds like an awesome life saver, more like the Superman of software engineering, but what really led to its development?

In the early 1990s, when the use of computers began to boom, and it was needed to have many different types of applications at the user's disposal. Computers were now being used in use cases that were never imagined before.

The waterfall model was just too slow to keep up with the pace at which businesses were moving - faster than they had ever moved in the entire history of computing. It meant that a business could change entirely over the course of three years, and this would spell doom for the project.

Many projects ended up being cancelled before they were finished, because they simply didn't meet the needs of the business anymore.

In industries such as aerospace, aviation and defence, projects could stretch on for as long as 20 years, and even beyond that if the system being developed was extremely complex.

(NEW SLIDE for example) A good example of this is the space shuttle that was operationally launched in 1982. The information that was used on this project, together with the processing technologies, was from way back in the 1960s! In today's terms, it would be unthinkable to develop any kind of software for that long. By the time you have finished, technology would have evolved and moved far beyond what you were developing, which will be a waste of time, money, and effort.

(NEW SLIDE for second example) This change was not only limited to computing. Other industries are also using agile development methods. In the 1990s, it took the automotive industry six or more years to design a new car from scratch. By the time we got to the 1990s, a whopping 50% or more was shaved off that development time!

The agile era

Frustrations with the linear models from yesteryear lead to the Utah meeting in early 2001. At this meeting, they looked at building software that adapts to any changes that come during the process. Rapid feedback, extensive engagement with the customer and being able to incorporate changes became the key features of software development. The idea was for the developers not to try and fully understand the project at first but rather to deliver a prototype of what thes understood it to be as quickly as possible, then adapt that prototype to the users' needs after receiving feedback.Unlike the old linear methods, the modern approach is that very little is set in stone at the beginning of the project. The focus here is to get to work as soon as possible.

Old vs new

Linear methods of old seek to set the budget, requirements, and scope at the very start of the project and in as much detail as possible. While that in itself is not such a bad idea, it quickly shows its weaknesses once you don't have time to attend to every little thing. Setting aside time to look at the requirements, costs, legal implications, availability of hardware and expertise and so on means that a lot of time will be spent doing these things and no actual development will be taking place. There really isn't anything tangible to show the customer until we are way beyond the start of the project, in fact towards the end!

If for some reason something in the project has to change, it means that all that research has to be done again! It will have to be seen how that little change will affect the project as a whole. This leads to further delays, which is why we say linear methods are not really built with changing requirements in mind.

Agile development on the other hand incorporates changes at every iteration of the product. There isn't really a final product until the customer is satisfied. There is something that is delivered to the customer at every milestone. Before agile development, the software development life cycle followed a very rigid process in which every step of the process had to be completed, dusted off and shelved before moving to the next.

While the actual original waterfall model could accommodate some changes, the reality was that budget and schedule almost always made this impossible. It typically forced teams to stick to earlier decisions and this had the tendency of disappointing the client. It was widely believed at that time that if you spend lots of time planning your code, you will spend far less time writing it. This only really made the process longer with the emphasis on the planning rather than on delivering actual work.

Agile programming principles

The agile development methodology is based on a number of principles, all of which place huge emphasis on the client.

Agile processes are particularly known for their lower cost. This is a result of spending less time gathering information as opposed to actually producing the solution. Because the project is very much transparent to the client, much less time is spent on the liberating requirements as there is constant feedback from the customer. It is also less likely to get the solution wrong because the client is extensively involved.

There are a number of reasons why the agile development process is taking over from where traditional methods left off. Let's look at each of them in detail

Quick and easy adaptation

It is extremely easy for teams to adapt to change when using agile processes. The nature of the process itself encourages teams to embrace and practise adaptable programming techniques. Angela acknowledges that customers typically change their requirements and team members should be able to adapt to these without going through a lengthy requirement change process, with lots of red tape such as review and approval.

Technical debt

This refers to the maintenance tasks that are required to support a product. In linear models, there's a huge amount of technical debt and this eats into development time. With agile development methods, any such documentation is added to the product backlog and attended to in each sprint. This is also used to determine what needs to be addressed next.

Minimising risk

The waterfall model comes with a level of uncertainty that it isn't quite addressed in planning. Testing is usually reserved to later stages, after all the code has been written and released. Since agile methods have functioning code at a very early stage, it is quite easy to go back and fix problems because there is a fair amount of feedback from early stages.

Higher quality

Because most of the time in the waterfall model is spent in planning, if for some reason the development team runs out of time, there is a rush to release features that are half baked. With agile development, features are not developed at the same time. Rather, small subsets of the features are developed in the sprints and developers have more time to perfect them before a release.

Predictable release dates

Because agile methods are built around shorter working times and the production of a tangible product at the end of each working period, it is easier to predict delivery dates. The client already knows that they will get new features at the end of every sprint. The waterfall model on the other hand relies on long cycles that do not produce results at first.

Collaboration and communication

The agile methodology is built around people working in teams and working together to produce the product. This means that there is good communication between the team members and the client. So there's a far less chance of surprises and unplanned changes to the project. There is also more control of the direction of the project and more focus on what the client wants.

Daily stand-up meetings keep everyone in the picture, and this means that any issues can be picked up and fixed fairly quickly.

More of what the agile development model entails is contained in the manifesto we covered in our previous lesson. Just to jog your memory, this table shows the key principles in the agile manifesto:

Principle	**Description**
Satisfaction and delivery	Customer satisfaction through early and continuous working software
Welcoming change	Welcome changing requirements, even at later stages of development
Deliver frequently	Deliver working software frequently (weekly rather than monthly)
Communication is key	Ensure close association of developers with businesspeople on a daily basis

Environment and trust	Build projects around motivated individuals, give them support and trust them
Face-to-face communication	Encourage face-to-face conversation to ensure efficient and effective communication
Software as measure of progress	Working software is the primary measure of progress
Sustainable development	Maintain a constant pace throughout the development
Attention to details	Continuous attention to technical excellence and good design
The power of less	Simplicity is essential
Self-organising teams	Team's attention is on being effective in changing circumstances

Agile frameworks

The idea of agile software development methodologies is built upon iterative development. One thing that is common among all these development methods is that at the end of each stage, there is a tangible delivery. It is also imperative to start work as soon as possible, and to deliver a working product even if it is not 100% perfect. This product is then improved by looking at it repeatedly, finding ways in which to do things better, and adding new features as required by the client. At the end of a set number of iterations, the customer is presented with a fully functional finished product, together with all the required documentation. Let's now look at some of the popular agile software development methodologies and see what they are all about.

Scrum

Scrum is a lightweight process framework which uses the agile methodology. This requires the overheads of the processes to be kept as small as possible so that time is allocated to getting useful work done. This is not to say that the time spent attending to overheads has been wasted. It just needs to be kept to a minimum because at the end of the day what we really want is a working software package. The scrum process is different from other agile processes because of its distinct steps.

Scrum is typically divided into three categories: roles, artefacts, and time boxes. It is clearly distinguishable from other methods due to these key features, that we will look at now.

Roles

There are three important roles in the scrum methodology. These are the product owner, the scrum master, and the team. Here's a table summarising these roles:

Product Owner	**Scrum Master**	**The Team**
● Defines features of the product	● Manages the team and looks after the team's productivity	● Approx 5-9 members
● Decides the release date and corresponding features	● Maintains the block list and removes barriers in development	● Includes developers, designers and sometimes testers, etc.

● Prioritises the features according to the market value and profitability of the product	● Coordinates with all roles and functions	● Organises and schedules their work on their own
● Responsible for the profitability of the product	● Shields team from external interferences	● Can do everything within the boundaries of the project to meet the sprint goal
● Can accept or reject work item result	● Invites to the daily scrum, sprint review and planning meetings	● Actively participate in daily ceremonies

Scrum artifacts

Scrum artifacts represent the work that provide transparencies and opportunities for inspection and adaptation. These are specifically designed with the goal of giving everyone the same understanding of what is going on in the project. Scrum artifacts constitute a number of activities, such as:

User stories: explanations of the functionality included in the system being tested. For example: “Users can pay by tapping a card” doesn't say much about how the system works but gives you an idea of what is included.

Product backlog: a collection of user stories prepared and approved by the product owner.

Release backlog: a list of the storeys which should be slotted for release. By release, we mean the period in which an iteration is completed. The idea is that all the storeys in the backlog are attended to and completed by the time we get to the next release.

Sprints: where the actual work in the release backlogs is done. These are typically decided by the product owner and the development team and can typically last anywhere between two and four weeks. During this time, the development team sets out to complete all the user stories in the product backlog.

Sprint backlog: almost like the release backlog, except that this is the list of user stories that are set to be completed in a sprint. This list is updated daily, and the work is not explicitly assigned. Rather, team members take up work on their own. A schedule of the work that remains at the end of the day is updated into the sprint backlog.

Block list: a list of decisions that haven't been made owned and updated by the scrum master.

Burn-down chart: the overall progress of work in the team and a reflection of the overall process. It is typically presented in a graph and shows the stories and features that the team hasn't gotten to yet. Partially completed users’ stories are also reflected in this graph.

CONFUSING - when we listed the three, we referred to TIME BOXES, now we are talking about CEREMONIES - are they the same thing???

The third aspect of the scrum is the ceremonies. These are actually exactly what they sound like - processes that result in milestones in the scrum process. They include:

Sprint planning: looking at the storeys from the release backlog and putting them in the sprint backlog. Meetings hosted by the scrum master in which testers estimate the time and effort needed to test the various storeys in the sprint backlog.

Daily scrum: a short meeting (max 15 mins) hosted by the scrum master to discuss what was delivered on the previous day, set the day's activities, and talk about what is planned for the next day. This is used as a tracking method for the team’s progress.

Sprint review: A 2-4 hour meeting hosted by the scrum master to discuss the accomplishments of the team and lessons learned in the past sprint.

You may have noticed that we have mentioned the scrum master quite a lot of times in the past a few minutes. In fact, it seems as though the scrum master is all we can talk about when we talked about ceremonies and artifacts. The scrum master is sort of an overseer on the project.

The scrum master ensures that all the team members follow the scrum's theories, rules, practices, and activities. The strong must make sure that goals in scope of the project are understood by everyone involved. It is also their responsibility to make sure that any obstacles are removed. In the same manner, they should also implement any changes that can improve productivity and speed of delivery. The most obvious duty is to facilitate scrum meetings and communicate with the product owner.

We've just mentioned another new person, the product owner. The scrum product owner is a key stakeholder in the project's development. They are responsible for managing product backlog and making sure that it is as clear as possible. Collaboration with the development team and stakeholders by gauging the team to achieve the best goals and missions is also a key responsibility.

Last but not least, we will look the obvious bit, the developers. The Development Team is comprised of developers proficient in their own area of expertise. Unlike the other Scrum Team members, the Development Teamwork on the actual implementation of the deliverable software which is to be delivered at the end of each Sprint.

The Development Team may consist of people having specialized skills

The Team is set up considering all the essential skill set required to successfully develop, test & deliver the product increments every Sprint without the outside help. This implies that the team is expected to be self-sufficient and cross-functional. The Development Team doesn't take any help from outside the Scrum Team and manages their own work.

The accountability of developing Increments always lies with the entire development team, but everyone in the Scrum Team is responsible for the overall delivery.

The development team has two main responsibilities.

Development and delivery: the development team creates all that is needed for and is part of the next scheduled release. It is the Product Owner's call to decide what needs to be part of the release.

Tasking and providing estimations: a big part of the development team's activities is picking up user stories from prioritised backlogs and delivering them in the next sprint.

The scrum process

The scrum process begins with the product owner, who creates a list of tasks to be done and what the final product needs to be like.

Next, the scrum team comes together for scrum planning, which has its emphasis on the first product backlog. The resulting list becomes the sprint backlog, which is then used in the sprint.

The sprint is a 2–4-week period that is punctuated by daily scrums and overseen by the scrum master.

At the end of each sprint, the sprint review meeting is organised by the product owner. During the meeting, the development team highlights what they have completed. In subsequent sprints, this will be highlighted against what they completed in the last sprint.

The product owner then gives information about what is remaining on the product backlog and estimated time to complete the project if needed.

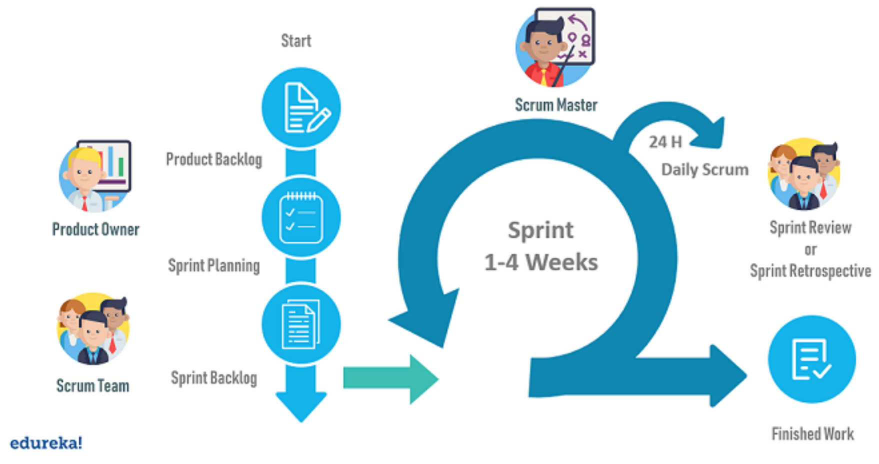
The cycle repeats for the remaining tasks in the product backlog.

This goes on until either of these things happen:

The deadline has been reached.

The budget is exhausted.

The product owner is satisfied with the final product.



Extreme programming

Extreme programming, which can be abbreviated to XP, is a lightweight, efficient, low-risk, flexible, predictable, and scientific way of building software.

It was developed judges the specific needs of software development in small teams that were faced by unclear and dynamic requirements. One of the expectations of extreme programming is that the team organises itself. It is built on practices that are simple and self-complete. The key assumption of extreme programming is that the cost of changing a program can be held constant over time.

It is built on a number of values that include:

- Emphasis on continuous feedback from the customer
- Short iterations
- Design and re-design
- Frequent testing and coding
- Early elimination of defects to reduce costs
- Active customer involvement
- Delivering a working product to the customer

You may be wondering why it is even called extreme programming.

This is because it literally takes effective principles and practises to extreme levels.

Code reviews are effective because code is reviewed all the time.

There is continuous testing and regression.

Integration testing is done several times a day.

Alterations are short.

Simply put, this is agile programming on steroids! It was formulated back in 1999 by Kent Beck, Ward Cunningham, and Ron Jeffries with the goal of improving agile methods.

Extreme programming has quite a lot of advantages. It has slipped schedules and achievable development cycles to ensure timely delivery. Testing is extensive and ongoing, and this ensures that any existing functionality will not be broken by new features. Also, making the customer part of the team means communication is as effective as can be.

All of this has the direct effect of producing software rapidly and ensuring immediate responsiveness to the customer's needs. There is a great focus on low defect rates which in turn brings higher customer satisfaction. This also reduces the overall project cost and boosts team cohesion and employee satisfaction.

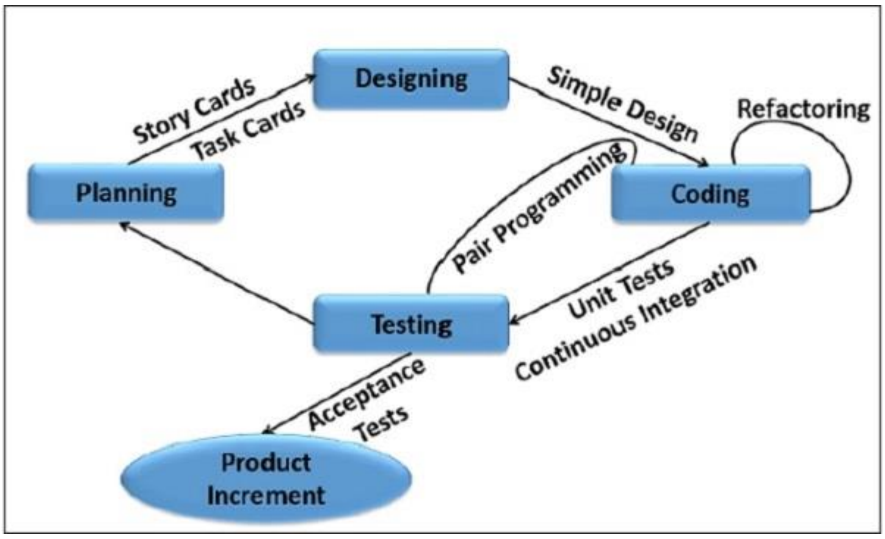
Extreme programming follows a pretty simple process.

Unit tests are written before programming and all these tests are kept running at all times. This is how defect that kept it in minimum.

From there, starting with a simple design, the team has just enough information to write code for the features at hand. Next, programming in pairs, which literally means two programmers on one screen, one programmer price the actual code and the other profiles reviews and other input. They both take turns to do each of these tasks.

This is followed by integrating and testing the whole system. This is done several times a day. The system, in its minimal state, is put into production, and the team immediately starts working on upgrades. The customer is involved in feedback. This feedback is used in the next iteration to build upgrades in the next iteration.

This diagram summarises the process:



Here we see that it is pretty much the same as add the agile methods except that most of the usual processes are done at a much faster pace and mostly concurrently. This is where the “extreme” part comes into play. Extreme programming has been known to produce software faster than any methods that we have looked at so far. The roles of each of the involved parties are pretty much the same as those we explained when we looked at scrum.

Kanban

Kanban is enormously prominent among today's agile and devops software teams, but the kanban methodology of work dates back more than 50 years.

In the late 1940s, Toyota began optimising its engineering processes based on the same model that supermarkets were using to stock their shelves. Supermarkets stock just enough product to meet consumer demand, a practice that optimises the flow between the supermarket and the consumer. Because inventory levels match consumption patterns, the supermarket gains significant efficiency in inventory management by decreasing the amount of excess stock it must hold at any given time. Meanwhile, the supermarket can still ensure that the given product a consumer needs is always in stock.

Kanban is based on a number of values, just like other frameworks. Teams applying Kanban to improve the services they deliver embrace the following values:

Transparency – sharing information openly using clear and straightforward language improves the flow of business value.

Balance – different aspects, viewpoints, and capabilities must be balanced in order to achieve effectiveness.

Collaboration – Kanban was created to improve the way people work together.

Customer focus – Kanban systems aim to optimise the flow of value to customers that are external from the system but may be internal or external to the organisation in which the system exists.

Flow – Work is a continuous or episodic flow of value.

Leadership – Leadership (the ability to inspire others to act via example, words, and reflection) is needed at all levels in order to realize continuous improvement and deliver value.

Understanding – Individual and organisational self-knowledge of the starting point is necessary to move forward and improve.

Agreement – Everyone involved with a system is committed to improvement and jointly moving toward goals while respecting and accommodating differences of opinion and approach.

Respect – Value, understand, and show consideration for people.

Practices

The following practices are essential to manage a kanban system.

Visualize

Kanban systems use mechanisms such as a kanban board to visualize work and the process it goes through. In order for the visualization to be the most effective, it should show:

- Where in the process a team working on a service agrees to do a specific work item (commitment point).
- Where the team delivers the work item to a customer (delivery point)

Policies that determine what work should exist in a particular stage

WIP Limits

Limit work in progress

When you establish limits to the amount of work you have in progress in a system and use those limits to guide when to start new items, you can smooth out the flow of work and reduce lead times, improve quality, and deliver more frequently.

Manage flow

The flow of work in a service should maximize value delivery, minimize lead times and be as predictable as possible. Teams use empirical control through transparency, inspection, and adaption in order to balance these potentially conflicting goals. A key aspect of managing flow is identifying and addressing bottlenecks and blockers.

Make policies explicit

Explicit policies help explain a process beyond just listing different stages in the workflow. Policies should be sparse, simple, well-defined, visible, always applied, and readily changeable by the people working on the service. Examples of policies include: WIP limits, capacity allocation, definition of done, and other rules for work items existing various stages in the process.

Implement feedback loops

Feedback loops are an essential element in any system looking to provide evolutionary change. The feedback loops used in Kanban are described in the lifecycle section.

Improve collaboratively, evolve experimentally

Kanban starts with the process as it currently exists and applies continuous and incremental improvement instead of trying to reach a predefined finished goal.

Roles

Given Kanban's approach to start with your existing process and evolve it, there are no roles explicitly called for when adopting Kanban. Use the roles you currently have on your team.

There are two roles that have emerged in practice that serve particular purposes. It's highly likely that these functions are filled by someone in an existing role as mentioned below.

Service Request Manager

Understands the needs and expectations of customers and facilitates the selection and ordering of work items at the replenishment meeting. This function is often filled by a product manager, product owner, or service manager.

Service Delivery Manager

Responsible for the flow of work to deliver select items to customers. Facilitates the Kanban meeting and delivery planning. Other names for this function include flow manager, delivery manager, or flow master.

Lifecycle

Because work items tend to flow through a Kanban system in single piece flow, and each system is different with respect to stages in its workflow, the best way to describe the lifecycle of the Kanban method is via the feedback loops involved.

Those feedback loops (cadences) are:

Strategy Review (Quarterly): select the services to provide and the context in which those services are appropriate.

Operations Review (Monthly): understand the balance between and across services, including deploying people and resources to maximize value delivery

Risk Review (Monthly): understand and respond to delivery risks in services

Service Delivery Review (Bi-Weekly): examine and improve the effectiveness of a service. This is similar to a retrospective that is focused on improving the Kanban system.

Replenishment Meeting (Weekly): identify items that the team will work on and determine which work items may be selected next. This is analogous to a planning meeting for a sprint or iteration.

The Kanban Meeting (Daily): a team working on a service coordinates their activities for the day. This is analogous to a daily stand-up.

Delivery Planning Meeting (Per Delivery Cadence): monitor and plan deliveries to customers.

Other models

The iterative process didn't just end with agile methods. There were other popular branches that sought to improve the agile process and evolved in their own way. Let's look at some of these briefly.

DevOps

DevOps is a collaboration between Development and IT Operations to make software production and deployment in an automated and repeatable way. DevOps helps to increase the organization's speed to deliver software applications and services. The word 'DevOps' is a combination of two words, 'Development' and 'Operations.'

DevOps is built on a number of key principles. Here, are six principles which are essential when adopting DevOps:

1. Customer-Centric Action: DevOps team must take customer-centric action for that they should constantly invest in products and services.

2. End-To-End Responsibility: The DevOps team need to provide performance support until they become end-of-life. This enhances the level of responsibility and the quality of the products engineered.

3. Continuous Improvement: DevOps culture focuses on continuous improvement to minimize waste. It continuously speeds up the improvement of product or services offered.

4. Automate everything: Automation is a vital principle of DevOps process. This is not only for the software development but also for the entire infrastructure landscape.

5. Work as one team: In the DevOps culture role of the designer, developer, and tester are already defined. All they needed to do is work as one team with complete collaboration.

6. Monitor and test everything: It is very important for DevOps team to have a robust monitoring and testing procedures.

DevOps allows Agile Development Teams to implement Continuous Integration and Continuous Delivery. This helps them to launch products faster into the market.

Other important reasons are:

1. Predictability: DevOps offers significantly lower failure rate of new releases.

2. Reproducibility: Version everything so that earlier versions can be restored anytime.

3. Maintainability: Effortless process of recovery in the event of a new release crashing or disabling the current system.

4. Time to market: DevOps reduces the time to market up to 50% through streamlined software delivery. This is particularly the case for digital and mobile applications.

5. Greater Quality: DevOps helps the team to provide improved quality of application development as it incorporates infrastructure issues.

6. Reduced Risk: DevOps incorporates security aspects in the software delivery lifecycle. It helps in reduction of defects across the lifecycle.

7. Resiliency: The operational state of the software system is more stable, secure, and changes are auditable.

8. Cost Efficiency: DevOps offers cost efficiency in the software development process which is always an aspiration of IT companies' management.

9. Breaks larger code base into small pieces: DevOps is based on the agile programming method. Therefore, it allows breaking larger code bases into smaller and manageable chunks.

DevOps Lifecycle

DevOps is deep integration between development and operations. Understanding DevOps Concepts is not possible without knowing the DevOps lifecycle.

Here is brief information about the Continuous DevOps life cycle:

1. Development

In this DevOps stage, the development of software takes place constantly. In this phase, the entire development process is separated into small development cycles. This benefits the DevOps team to speed up software development and the delivery process.

2. Testing

The QA team use tools like Selenium to identify and fix bugs in the new piece of code.

3. Integration

In this stage, new functionality is integrated with the prevailing code, and testing takes place. Continuous development is only possible due to continuous integration and testing.

4. Deployment

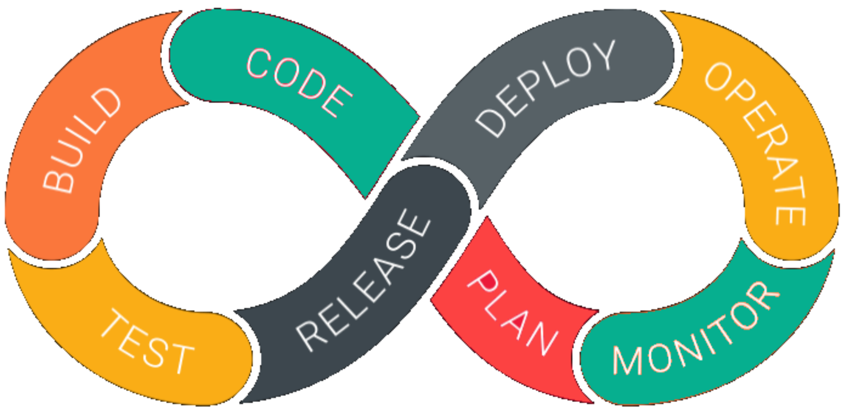
In this phase, the deployment process takes place continuously. It is performed in such a manner that any changes made any time in the code, should not affect the functioning of high traffic website.

5. Monitoring

In this phase, the operation team will take care of the inappropriate system behaviour or bugs which are found in production.

DevOps Workflow

Workflows provide a visual overview of the sequence in which input is provided. It also talks about actions performed, and output generated for an operations process.



DevOps Workflow

Workflow allows the ability to separate and arrange jobs which are top requested by the users. It also gives the ability to mirror their ideal process in the configuration jobs.

Feature-driven development (FDD)

FDD is an agile, highly adaptive software development process.

It: Is highly and short iterative.

Emphasizes quality at all steps.

Delivers frequent, tangible working results at all steps.

Provides accurate and meaningful progress and status information, with the minimum possible overhead and disruption for the developers.

Is favoured by client, managers, and developers.

FDD decomposes the entire problem domain into tiny problems, which can be solved in a small period of time, usually 2 weeks. decomposed problems are independent from each other, and this reduces the need for communication. In FDD, the concept of quality is broadened so as not just to test the code, but also include things such as coding standards, measuring audits and metrics in the code.

FDD defines six key roles and implies a number of others:

The Project Manager (PM) is the administrative head of the project responsible for reporting progress, managing budgets, fighting for headcount, and managing equipment, space, and resources, etc.

The Chief Architect (CA) is responsible for the overall design of the system. He/she is responsible for running the workshop design sessions where the team collaborates in the design of the system. The work requires both excellent technical and modelling skills as well as good

facilitation skills. He or she steers the project through the technical obstacles confronting the

project.

The Development Manager (DM) is responsible for leading the day-to-day development activities. In a facilitating role requiring good technical skills, the Development Manager is responsible for resolving everyday conflicts for resources when the Chief Programmers cannot do on their own.

The Chief Programmers are experienced developers who have been through the entire software development lifecycle a few times. They participate in the high-level requirements analysis and

design activities of the project and are responsible for leading small teams of three to six developers through low level analysis, design, and development of the new software's features.

The Class Owners are developers who work as members of small development teams under the guidance of a Chief Programmer to design, code, test, and document the features required by the

new software system.

The Domain Experts are users, sponsors, business analysts, or any mix of these. They are the knowledge base that the developers rely on to enable them to deliver the correct system. Domain Experts need good verbal written and presentation skills. Their knowledge and participation are absolutely critical to the success of the system being built.

Supporting Roles:

Domain Manager

Release Manager

Language Guru

Build Engineer

Tool smith

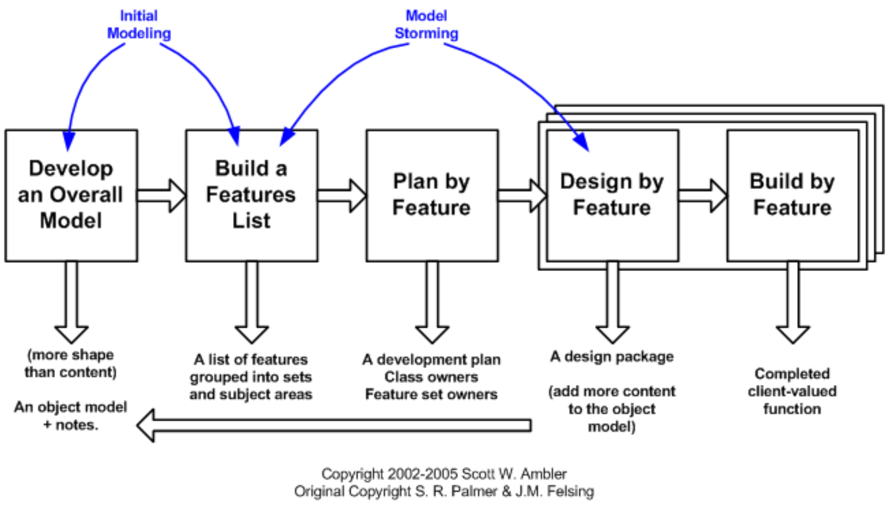
System Administrator

Testers

Developers

Technical writers

This diagram summarises the process that FDD follows:


Initial Modeling
Model Storming
Develop an Overall Model
Build a Features List
Plan by Feature
Design by Feature
Build by Feature
(more shape than content)
An object model + notes.
A list of features grouped into sets and subject areas
A development plan Class owners Feature set owners
A design package
(add more content to the object model)
Completed client-valued function
Copyright 2002-2005 Scott W. Ambler
Original Copyright S. R. Palmer & J.M. Felsing

FDD's five steps are supported by several practices. The first is domain object modelling, the creation of a high-level class diagram and supporting artifacts that describes the problem domain. Developing by feature and individual class ownership are also good practices, as is having developers work together in feature teams. Inspections are an important aspect of FDD.

FDD also insists on regular builds, similar to XP, and configuration management. Finally, FDD promotes a best practice called reporting/visibility of results, similar to XP and AM's philosophy of open and honest communication.

Conclusion

With that, we come to the end of our lesson. Software development is a pretty broad topic and can be an entire course on its own. We just looked at the basics that you need to get started in your quest to become an awesome programmer. We still have a little more that we need to cover, so let’s make a date for our next lesson. That’s all from me, see you next time!

References

Alliance |. (2017). What is Kanban? [online] Available at: https://www.agilealliance.org/glossary/kanban/#q=~(infinite~false~filters~(postType~(~'page~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'kanban))~searchTerm~'~sort~false~sortDirection~'asc~page~1)

Agilemodeling.com. (2020). Feature Driven Development (FDD) and Agile Modeling. [online] Available at: http://agilemodeling.com/essays/fdd.htm

Archana Choudary (2019). Agile Scrum Tutorial | How to Develop a Product Using Scrum? | Edureka. [online] Edureka. Available at: https://www.edureka.co/blog/agile-scrum-tutorial/

Atlassian (2019). What is kanban? [online] Atlassian. Available at: https://www.atlassian.com/agile/kanban

Cprime. (2019). What is AGILE? - What is SCRUM? - Agile FAQ’s | Cprime. [online] Available at: https://www.cprime.com/resources/what-is-agile-what-is-scrum/

Extreme Programming - The Agile Methodologies (2021). Extreme Programming - The Agile Methodologies. [online] Umsl.edu. Available at: https://www.umsl.edu/~sauterv/analysis/Fall2013Papers/Buric/agile-methodologies/extreme-programming.html

Rungta, K. (2020). DevOps Tutorial for Beginners: Learn Now (Training Course). [online] Guru99.com. Available at: https://www.guru99.com/devops-tutorial.html

Scrum.org. (2020). What is Scrum? [online] Available at: https://www.scrum.org/resources/what-is-scrum

Simplilearn.com. (2019). Agile Scrum Introduction Tutorial | Simplilearn. [online] Available at: https://www.simplilearn.com/agile-scrum-introduction-tutorial-video

Softwaretestinghelp.com. (2021). Scrum Team Roles and Responsibilities: Scrum Master and Product Owner. [online] Available at: https://www.softwaretestinghelp.com/scrum-roles-responsibilities/

VC Web Design. (2013). What is Agile Methodology? and Why Do We Use It? - VC Web Design. [online] Available at: http://vcwebdesign.com/uncategorized/what-agile-methodology-why-do-we-use-it