FALMOUTH
UNIVERSITY

GAMES
ACADEMY

Agile development: from theory to professional practice

2019-20

Ver 0.1: July 25th 2019

Contents

Introduction

On the course, we've looked at several different flavours of agile development, including **scrum**, **extreme programming** and **Kanban**.

Whilst these approaches to agile all have different names and define their activities and artefacts differently, they all tend to embody at their heart the agile 4-stage process (**Plan**, **Do**, **Test** and **Review**).

Agile offers a lot of potential advantages to more traditional '**waterfall**' development.

During each iteration (loop) of development, because the application is tested and reviewed giving the project stakeholders an opportunity to correct drift in the project fairly early on, rather than at the end of months of development. This makes the the 'agile' process aptly named.

It's worth remembering that agile is often used in development situations where a team of developers is working with a client and much of the final say of what the application in development does is down to the client rather than the developers.

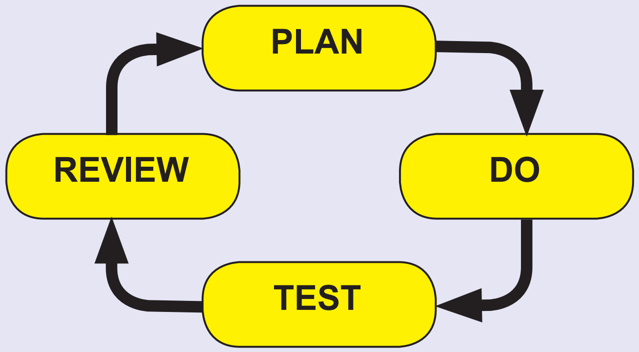


Fig 1. Agile stages

- **Plan -** Focuses on creating a plan of work that can be undertaken with the **sprint period** by the team of developers assigned to the project. This gives rise to the creation of **User Stories** that articulate what functionality the developed system will achieve, usually from the position of the end user, typically expressed as 'as a player I want to …'.
- **Do -** The user stories will give rise to a **task list** for the sprint from which the do phase of the sprint is conducted, with developers working together to complete tasks with progress, and blocking issues, reported through the periodic (daily) team sprints.
- **Test -** At the end of the development phase, the application is tested
- **Review -** with the results leading into the review phase that is used to determine the phase (iteration) of development.

Contents

Agile for creative game development

For a lot of game development projects (and in particular, those developed in the Games Academy) the development team are the clients, in the sense that they have creative control. In these situations, we need to think about how agile development processes can work for us and the games we develop.

A lot of game development is concerned with risk reduction and answering two fundamental questions:

- **Can we implement a feature?**
- **Should we implement a feature?**

You can think of this a bit like the scientist in Jurassic Park but without the ethical overtones. The 'can we implement a feature' question is asking if it is possible to add a particular feature to the game, for example 'can we accurately simulate the entire population of the Earth', or 'can we use subsurface scattering on all our in-game objects'. Work needs to be undertaken to see if these features can be added to the game and what kind of cost that adds in terms of:

1. **Development time** - *the time it will take artists to create models.*
2. **Run time** *(the frame time it takes to update millions of AI agents per game turn).*

.This effectively means that a lot of the development process is about asking questions that prototypes (and **user testing**) will address. This gives us a very different perspective on the agile process. Rather than looking at the 'plan' stage as the starting point for each iteration, we need to look at 'test' as the starting point.

The process is now better considered as:

1. *What risks and concerns do we want to address this iteration?*
2. *What tests do we need to create to address those risks and concerns?*
3. *Build a plan based on those tests versus the amount of people you have to do the work*
4. *Undertake the sprint development work*
5. *Performing testing*
6. *Use the results of testing to address current risks and concerns and to highlight new risks and concerns*
7. *Go to 1 (ie. repeat and reiterate the process)*

Let's have a look at a couple of development examples to see how this works (see opposite).

Can we implement a feature?

To a large degree, the first question is about R&D and can be factored into agile sprints as a yes/no testing task - can we actually implement this and are the costs (development & run time) acceptable.

Should we implement a feature?

This question is asking if adding a particular feature to the game will make it a better experience for the player and can be thought of as 'is this fun'. This is a much harder question to answer because it relies on user feedback. To a degree, as developers, we can answer this question as a proxy for the players, but the most accurate feedback will come from the players themselves - hence the importance of user testing in game development.

Example 1

A team is developing a platform game.

Functionally, the game works, but play testers regard the game as frustrating to play and they don't want to play the game more than they have to. The team decides to evaluate what makes the game fun by testing different level and baddie combinations as part of a larger playtest session, effectively creating a large A/B test and interpreting the results.

The plan of work is to make several small levels with different features (number and types of collectibles in a level, different routes through a level, hidden and secret things, baddie type and placement) and different attributes for player and baddies (character speed, jumping abilities, baddie aggressiveness and so on). These levels are then tested with the results recorded to see which levels get played the most and which levels are reported to be the most enjoyable.

After running the user tests, the team goes through the results to discover that the play tester had enjoyed some of the features the designers thought they would do, but had really enjoyed some features that no-one thought they would like.

This feedback was used in the next iteration planning phase to build levels containing these sets of features to test if that was really what the players enjoyed.

Example 2

A team is developing a 2D shoot-em up game for mobile.

The game works and has been well-received by play testers, but the team feels the game looks 'flat' and 'dull' and would like to undertake an art pass to make the content better. One of the team's programmers and an artist are keen to experiment with a different rendering approach that will add sparkles.

This work can be done and then evaluated to see:

1. *The performance cost of doing it on low-end machines and*
2. *The impact it has on players.*

After testing the new graphics, only 10% of play testers notice the difference and half of those that do think the game feels worse than it did before. The feature is abandoned.

Feature-based development

From the examples given on the previous page, it's worth thinking about each sprint of development as a collection of activities that are undertaken by the team as a whole, but the whole team doesn't have to partake in each activity.

Typically, agile game development teams will form micro or **feature teams** that may last for a very short time (1 day) or may run across multiple sprints. The philosophy behind this approach is to make the feature team responsible for delivering a feature rather than just viewing a feature as something where different disciplines will just 'do their bit' and then throw the work in progress 'over the fence' to the next discipline.

For the micro team in 'Example 2' developing a new rendering approach for their game, it's easy to think of the artist creating a set of artworks and leaving it for the programmer to implement. However, we know (from bitter experience) that game development is an iterative activity, so it's likely that whatever you make will need to be finessed and refined to make it work in game. In this case, **the artist and programmer should work together** adding and refining content and algorithms until the feature is working as expected or as best as they can make it work.

For work that's likely to involve more disciplines, for example creating the player character for a 3rd person game, the feature team is likely to pull in a lot of disciplines: artists to create the character, animators to animate it, a programmer to make it respond to the player's inputs, a designer to make it behave properly and an audio specialist to create the right sounds for the character's behaviour.
We could look at this as a production line, where each feature team member 'does their bit' and then moves on to their next task, however, we can see from experience there's a lot of interaction between disciplines here and the player character feature will need a lot of refinement to firstly make it work well-enough to be signed off by the feature team (the 'can this be done' question) for testing and to then be play tested (the 'is this fun question').

Branched-based development

Up until now, there's an assumption that the whole development team is working on the same plan-do-test-review programme, but this doesn't have to be the case. For example, testing can occur before all the work in the do phase has been completed and, often, testing results can feed back into the do phase of development.

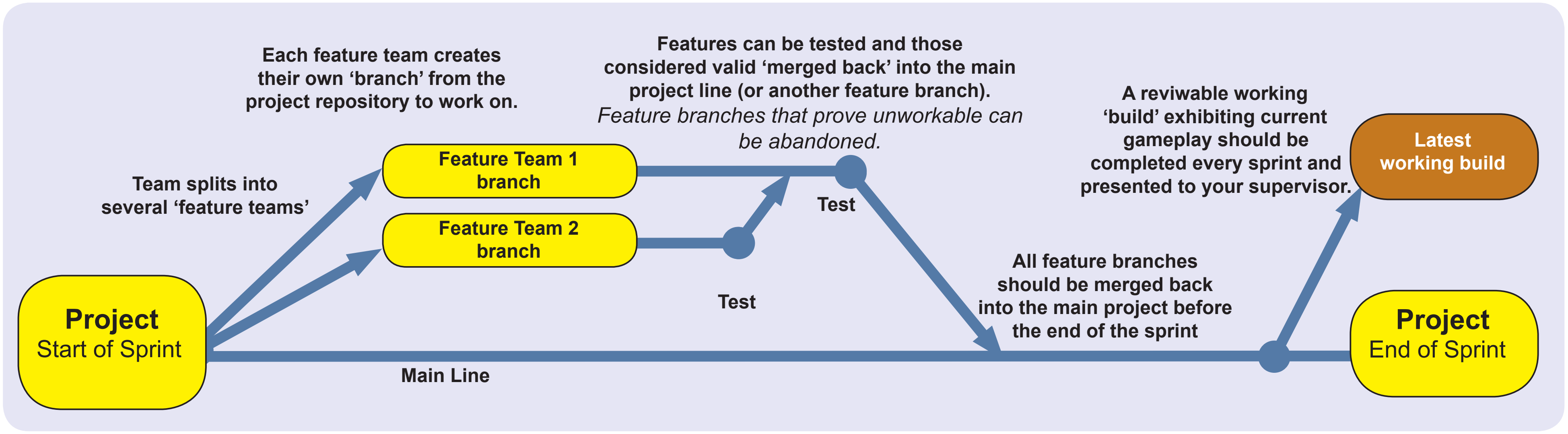
Of course, trying to manage this when everyone is working on the same project is likely to be very difficult which is why most **version control systems** (see boxout) support branching. Let's have a look to see how this can work for a project.

A team is developing a game and for a particular sprint is going to run as 3 feature teams developing features than can be developed and tested in isolation. Each feature team creates a branch from the repository and works on their feature using that branch. Each branch is a copy of the main line of the repo and can be treated like a normal git repo, with members of each feature team getting and pushing their work which can be shared by the feature team members.

As each feature team gets into a position to user test, they can do this in isolation from the main line of the project. Assuming user testing goes well and the feature is going to become part of the actual project it can then be integrated into the main line of the project and tested (to make sure the game still works) before being committed. That way, the main line of the project repo should always contain a working game.

If user testing results in the feature being abandoned (or shelved) the feature branch can be left as is. If developers need to work across multiple branches, GIT (see Version Control Systems opposite) will handle that context switching and no work in progress should be lost or copied into the main line, or another branch, by mistake.

NOTE THAT:
Branching sprints should aim to have working builds at the end of EVERY sprint. It is NOT acceptable to leave branches unmerged over multiple sprints as this stores up problems for the future and means that the build is not representative of progress.



Each feature team creates their own 'branch' from the project repository to work on.
Features can be tested and those considered valid 'merged back' into the main project line (or another feature branch).
Feature branches that prove unworkable can be abandoned.
A reviwable working 'build' exhibiting current gameplay should be completed every sprint and presented to your supervisor.
Latest working build
Team splits into several 'feature teams'
Feature Team 1 branch
Feature Team 2 branch
Test
Test
All feature branches should be merged back into the main project before the end of the sprint
Project Start of Sprint
Main Line
Project End of Sprint

Agile development for group projects in the Games Academy

The Games Academy has several modules of group game development that are geared around using agile development process and assessed based on your ability to perform as part of an agile team. To help with this, each agile team has a member of staff as a '**project supervisor**' to whom your team will present their plans and progress in weekly project meetings. These weekly meetings will alternate between:

- **A planning/review meeting** at the beginning/end of each two-week sprint
- **An individual progress/development meeting** mid-way during the sprint.
- There will also be an '**initial meeting'** held at the start of the study block to discuss the project in general and start the process off.

Module specific Assessment

There is a great deal of similarity between each module on how agile team-working practices are assessed and this is reflected here.

However, you should look at the assignment details for your module to ensure that you deliver all that ius required.

Initial Meeting

These initial meetings will vary by module, but overall should allow the team and supervisor to discuss the project in detail and allow for a discussion on the originality, and practicality of the game concept/idea as a viable team project.

This is also an opportunity to discuss if the team have the required resources, skills and manpower to realise their original concept.

Defining sprint tasks

Defining sprint tasks is hard, but gets better with practice.

When defining your tasks:

- Break down your tasks into sub-tasks that are no longer than 2-4 hours. If you can't do this, then you don't understand it well enough and you need to make it a task to gain that understanding.
- Estimate your task durations in hours - you will get better at this.
- Consider dependencies and who needs your work to get their work done.

Planning / review meeting

This meeting is split into two parts and will effective cover the **review** and **plan** parts of the agile cycle.

Review part

The team should initially present their progress review (which should take around 30-40mins). The team will present the outcome of their sprint as a feature-by-feature presentation, with each feature team being responsible for outlining the goals of their work, the outcomes and what the results mean for the game and future development.

Given that much of development is geared around answering the two development questions of can we do this and is this fun, it's perfectly acceptable for the outcomes to be 'no'. However, the team and feature team should ensure to get to a point where they can answer their questions and look to rescope their work if they are likely to run out of time in the sprint.

The review should feature game demos which can be a combination of feature teamwork and the current game as it stands. For all teams there should always be a working build of the game and making standalone builds should be part of the team's work plan. Teams are strongly encouraged to present all of their demo work (feature builds and game builds) as standalone builds rather than as 'in-editor' experiences, given the performance characteristics of editor demos.

Planning part

The remaining time should be used to present their plan for the next iteration of development. It's expected that the team will present a clear and defined plan which is likely to only require small edits as a result of the review meeting.

Team's present their plan of work as a sprint, outlining the activities that are to be undertaken, in terms of the feature teams, their make-up and what testing will be undertaken to determine the success or otherwise of the feature.

Typically, each feature will consist of a briefing document which will include appropriate user stories that are either:

- **Inward** ('as an artist/designer/ programmer I want to …')
- **Outward** ('as a player I want to …') which are broken down into appropriate (well-defined and estimated) tasks and put into the backlog as work to be done.

Ideally, briefs, user stories and tasks will take less than 10 days of linear effort, i.e. work should be completed within a sprint. Any work that will take longer than this should be broken down into smaller briefs so it will fit within a sprint.

Mid-sprint review

The mid-sprint review is an opportunity to see how development is progressing compared to the original plan from the planning meeting. This will be an opportunity to highlight issues and achievements and address problems before they become significant issues.

Individual progress

This is a chance for all of the team members to explore and develop their skills as agile practitioners. Each team member will fill in performance forms online to assess their teammates' performance (and their own). These results are then explored during the meeting.

Although these self and peer assessments will not dictate any final grade, they will nevertheless be used as a way of informing a team's supervisor of the overall involvement and engagement of a student in both the project and teamworking.

Given the personal and potentially critical nature of personal feedback, everyone is reminded to be civil with their feedback as part of your own performance assessment is drawn from these sessions. Therefore, as a team member, avoid saying things like 'you are a terrible person and I hate working with you' and prefer things like 'I find it hard to work with you, it would help me if you could attend the supervised studio practice sessions'.

Major Reviews

In year long team projects (such as GAM220/GAM240 or GAM320/GAM340) Week 8 of Study Block 1 and week 4 of Study Block 2 will also look critically and in depth at each project's overall progress, feasibility and scope, to help prevent projects going off track / becoming unviable.

Projects that fail these reviews may have to undertake major changes, or even (in the worst cases)reconsider the entire project from scratch.

How much time should a group project take?

Given your study commitments, it may not be possible to have stand-ups every day and it may not be possible to commit to working on the group project every day. The minimum commitment to the group project should be at least the period(s) of the supervised studio practice that you have for your associated module (e.g. GAM130, GAM220, GAM240, GAM320 & GAM330).

You are of course also expected to work on these projects outside this period in your own time, though this amount varies depending on your course structure. If in doubt talk to your project supervisor and/ or personal tutor about this and this can then be made clear to other team members during mid-sprint reviews.

Day to day development work (and Standups)

Agile development is typically defined through its daily stand-ups, a period in the day where the development team comes together for a short sitrep meeting, with team members reporting what they are working on and any blocking issues that they are facing.

Any issues can then be addressed in a follow-on meeting to keep the stand-up meeting as short as possible.

The best time for Standups

Agile practitioners argue about when the stand-up meetings should occur, some say first thing in the morning, others after lunch and others at the end of the day.

As a development team, it's your responsibility to work out the most suitable time for your stand-ups and to ensure that you actually have them with all the team members present (virtually or physically).

Development work is undertaken with feature teams, so it's helpful if the team can work together and be co-located as this work tends to ping-pong between team members.

Likewise, it is also advantageous if feature team members can commit to similar levels of effort and availability for their shared work as being blocked by an absent team member does not help team morale and motivation.

Dealing with conflict

Conflict is completely normal in any activities where people come together with different ideas and isn't something to shy away from. The challenge for development teams is to be able to separate people from conflicts that they are involved with and to 'play the ball not the player'. Typically, this occurs when conflicts break down into personal insults which causes a lot of upset and will often lead to long term resentments. The individual review meetings are a good case in point where you need to develop strategies to give people honest feedback without causing personal grief.

Typically, in creative projects, conflict will occur with what people want to make, how they want to make it and what looks 'good' or 'done'. A good way to address these issues is to rely on the group as the arbiter of decision making and for everyone in the team to accept that you don't always get what you want in development and not every hill is worth dying for.

Let's suppose on a project a feature has been implemented and you think it's rubbish, but the rest of the team really likes it. Unfortunately, you are going to have to suck that up as the team is more than just you. Conversely, a feature has been developed and some of the team like it, but most of the team don't, then the team has spoken, and you go with the majority.

If you reach a complete impasse, talk to your supervisor or toss a coin.

Resisting the urge to re-develop another's content

During any development project, everyone wants to make the best game that they can. However, often different people have different definitions of what good looks like and it can be very easy to look at an asset (model, texture, animation, class etc) that doesn't work well and look to rework it or replace it as it's quicker for you to do it than to get the original author to do it.

Whilst this can be really tempting, it generally has a bad effect on the team and the person who has had their work replaced and can lead to the withdrawal of effort and goodwill on projects. A better approach is to help the original author to rework their assets as this has the twin benefits of keeping them fairly happy with the project and helping them to develop their skills. It's worth remembering that as students you're all learning, so if you can share your expertise in an area you are strong in, that's likely to get paid back with you upskilling in another area.