

Ass1

by Adam Benjamin Howells

FILE	25301_ADAM_BENJAMIN_HOWELLS_ASS1_302864_1322570468.DOCX (1.71M)		
TIME SUBMITTED	04-DEC-2020 11:50PM (UTC+0000)	WORD COUNT	3943
SUBMISSION ID	138392867	CHARACTER COUNT	20402

Adam Howells

Student Name: Adam Howells (1707782)

Lecturer Name: Michael Dacey



09/11/20

Assignment 1

(ACCA7001) Agile Software Engineering Principles



TABLE OF CONTENTS

1	Introduction	3
2	Procedure	3
3	Background.....	4
3.1	Agile Methodology	4
3.2	Scrum Framework	5
3.3	Why Scrum?	7
4	Evaluation of Agile at the DVLA	8
4.1	Agile Practices and Tools	8
4.1.1	User Stories.....	9
4.1.2	Estimation	13
4.1.3	Product Backlogs	14
4.1.4	Sprint Backlogs.....	15
4.1.5	Collaboration	16
4.2	Continuous Integration and Testing Processes	18
5	Bibliography	21



1 Introduction

The Agile development methodology is quickly becoming an industry-standard in software development [1]. This report will first outline the general principles and values of Agile as well as the Scrum, a framework used with Agile.

Once Agile and Scrum has been established, an evaluation of the Agile development practices and tools used at the DVLA will be carried out. This will include agile methods such as user stories, estimation, backlogs, and how developers collaborate on projects. Each piece of discussion will be supported by real-world examples along with appropriate descriptions and analyses. Finally, a reflection will be carried out on the importance of continuous integration and testing processes which will include overviews of some technologies and techniques used at the DVLA.

2 Procedure

To tackle this assignment, a variety of resources will be used as references to support and justify the content. When describing Agile, the Agile Manifesto [2] will be a core reference, while the majority of references will come from journal articles available on various online libraries such as IEEE, and books available on the UWTSD online library. For evaluating the DVLA approach to agile, resources from DVLA's intranet will be used.



3 Background

3.1 Agile Methodology

The agile methodology was first conceived by the Agile Alliance, a group of software developers who shared similar goals and values; "mutual trust and respect, promoting collaborative, people-focused organizational models, and building the types of professional communities" [2]. The produced document, the Agile Manifesto, included four overarching values and twelve principles that built towards an ultimate goal; to quickly respond to customer needs and market changes.

More Valuable		Less Valuable
Individuals and Interactions	over	Process and tools
Working software		Comprehensive documentation
Customer Collaboration		Contract negotiation
Responding to change		Following a plan

Table 1: Agile values. Source: <https://agilemanifesto.org/>

By prioritising interactions between people over a strict development process, agile allows teams to build close relationships and a close working environment to boost team spirit and morale [3]. This allows teams to not be tied down to a specific process which may restrict full potential. By building close relationships, teams can get a good grasp on the ability and skill of other teammates which enables them to assign members to tasks that can complete it within the given timeframe.

The second value of prioritising working software is very important. Continuously producing working software is how value is measured during projects. The keyword is *continuous*. By releasing working software at regular intervals, each iteration of the software is kept simple. This lessens the need for comprehensive documentation.

In traditional methodologies such as waterfall, customer input is usually taken before the project begins as part of the contract negotiations. There's a risk that the customer doesn't know exactly what they want, or they change their minds, or the team can't create exactly what they want due to changing circumstances. By collaborating with the customer at each stage, customer feedback can be used more frequently to ensure that the contract is fulfilled. It's a lot easier to change something by customer request during development than it is to change it after the project is finalised.

Finally, participants in the agile process shouldn't be afraid of adjusting a plan if the need for it arises. It's impossible to fully predict setbacks before development begins, and customer collaboration means features will almost certainly be edited on feedback.



Contracts are typically formed with tools which allow changes and enhancements to be made during development [3].

Keep in mind, these are values which are preferred. It does not mean that you shouldn't write documentation or follow a plan. The Agile Alliance recognizes these values as being important, it's just a matter of "What do you value more?" [2]. Regarding contract negotiations, Abrahamsson et al. [3] say "the importance of well drafted contracts does grow at the same pace as the size of the software project".

3.2 Scrum Framework

According to Schwaber and Sutherland [4], Scrum is a lightweight framework first developed in the 1990s, before agile was defined. Its goal is to help "teams and organizations generate value through adaptive solutions for complex problems". With this, they outline five values that scrum is dependent on commitment, focus, openness, respect, and courage.

In 2002, Schwaber and Beedle [5] defined the various roles and responsibilities found in Scrum. There are three major roles:

- **Scrum Master** – ensures that the management of a project is following scrum values and practices. They also ensure that the project runs smoothly, where any impediments are handled with to maintain productivity. They communicate with the team and product owner.
- **Scrum Team** – the project team which are involved with carrying out tasks. They have the authority to organise themselves and make decisions to achieve the goals in a sprint. They also are involved with estimation and the creation of the sprint backlog.
- **Product Owner** – Chosen by the scrum master, they are responsible for the project. They manage and control the product backlog, and has the final say in the tasks added to the backlog. They also participate in the estimation of the effort of tasks in the backlog.

To combat chaos caused by unpredictability and complexity, Schwaber and Beedle [5] also outline a set of practices to be used (there are no requirements to use specific development methods):

- **Product Backlog** – this is a backlog of everything that needs to be completed for the product to be final. It's not ordered by priority and is updated constantly during the project. Examples of items that are added to a product backlog are features/functions, enhancements/upgrades, and bug fixes. Many roles can be involved with the creation of the product backlog.
- **Estimation (effort)** – this is the iterative process of taking items from the product backlog and predicting the effort required to complete. The unit of effort is



typically the number of hours a person needs. The more information that is available on the item, the more accurate the prediction comes. The product owner and scrum team are responsible for this.

- **Sprint** – a specified timescale of work. They are typically 1-4 weeks. The reason for having segmented periods of work is to allow for adapting to changing variables in between sprints, which may include the product requirements or technologies changing. When a sprint begins, items from the product backlog are moved into the sprint backlog. The general goal is to complete the sprint backlog within the sprint. Working tools in a sprint include:
 - **Sprint Planning Meetings** – a meeting is held at the creation of each sprint which is organised by the scrum master. Participants typically include the product owner, scrum team, and customers. The goal of the meeting is to decide what the goal of the sprint is, and what items from the product backlog should become a part of the sprint backlog.
 - **Daily Scrum Meeting** – before each working day, a daily meeting is held involving the scrum team and scrum master. These meetings are intended to be used to keep track of progress and are intentionally kept short and concise, typically lasting 15 minutes. Each member of the scrum team aims to answer three questions; what you did yesterday, what are you going to do today, and were there any impediments or deficiencies encountered.
 - **Sprint Review** – these are organised on the final day of the sprint. The scrum team and scrum master present the working software increment to the product owner and customer. This provides an opportunity to decide on the future and may introduce new or change items in the product backlog.

With the goal, values, roles, and practices defined it's time to look at the Scrum process itself. It accomplishes the goal of generating value through solutions for problems by [4]:

1. The product owner outlines the work required for a problem to be solved and is added to the product backlog.
2. A selection of the product backlog is turned into an increment of value during a sprint by the Scrum team.
3. A sprint review involving the Scrum team and any stakeholders are held to evaluate results, with adjustments made for the next sprint.
4. The process is repeated

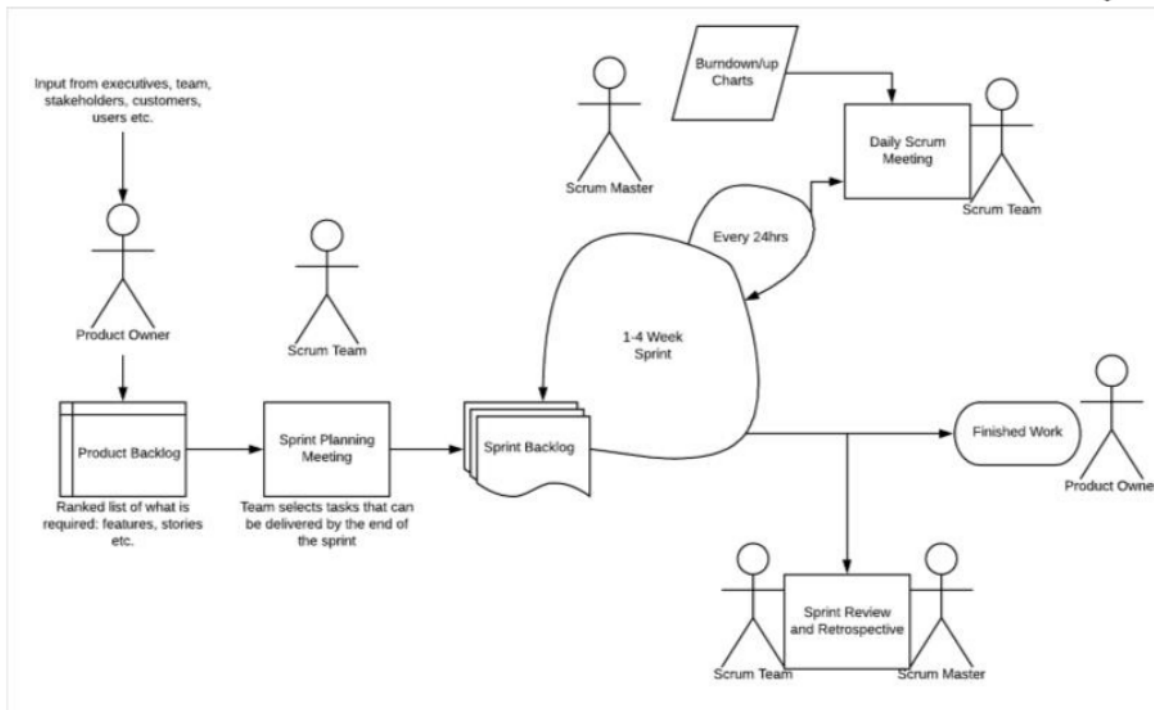


Figure 1: A detailed diagram showing the Scrum process.

3.3 Why Scrum?

Srivastava et al. [6] say that scrum is currently "the top most technique used in development not only for software but even in the fields of finance, research etc". Why is scrum the most popular framework for agile? For starters, scrum provides a very easy way to implement the agile methodology. Many empirical studies have proven that scrum is also the best performing framework. For example, Hayat et al. [1] researched businesses using various agile frameworks and compared Scrum, Crystal, Feature Driven Development (FDD), and Dynamic Software Development Methodology (DSDM). Over 52% uses scrum, with 95.2% agrees that scrum has had a positive influence on software project management (SPM). For time and cost-effectiveness specifically, 60% uses scrum. The survey results indicate that scrum reduces risk, cost, and time needed to deliver a quality product.



4 Evaluation of Agile at the DVLA

I was allowed to be a part of a scrum team for a trial project. The project was the Trade Licensing Service (TLS), which is a system that allows users to manage and renew trade licenses online as currently, it is all paper-based. The service involves two components: the internal and external. The external, built with Ruby on Rails, allows users to manage licences themselves via "gov.uk". The internal, built with React.js, allows employees at the DVLA to manage user's licence over the phone in case they don't have access to the internet.

4.1 Agile Practices and Tools

From what was understood of agile and scrum practices in the research, being a part of a project at the DVLA felt very familiar. There wasn't anything that was done differently which contradicted the literature. The only exception is the service designer being the product owner, which is due to the DVLA being in the public sector. The main thing that is DVLA-specific is the tools and technologies used to implement these practices.

The most essential software used is the Atlassian suite. This is a collection of software used by project managers and software developers [7]. Jira is used for project management and features a Kanban interface with DevOps tools which allows users to plan, track, release, and monitor projects [8]. Other features include product backlog, sprints, reports and charts.

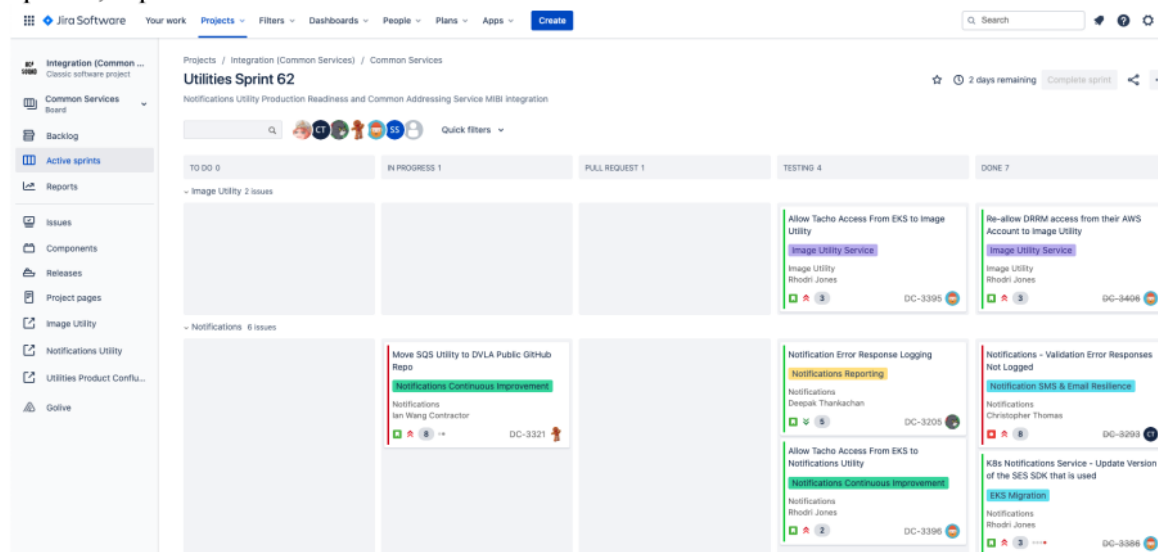


Figure 2: Screenshot of a project's Kanban board in Jira.

Another Atlassian product used is Bitbucket. This is used for git code management and allows for teams to collaborate on code, testing, and deployment [9]. It also has built-in continuous integration/delivery (CI/CD). The justification for using Bitbucket over other



git managements such as GitHub is the ability to have seamless integration with Atlassian products such as Jira.

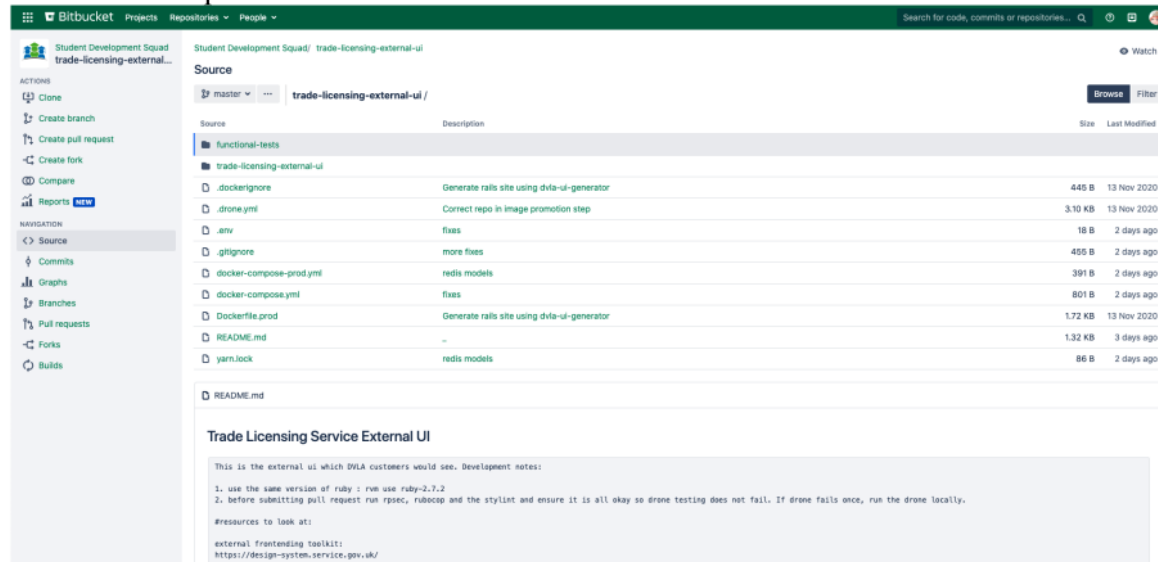


Figure 3: Screenshot of the source code interface of a repository.

4.1.1 User Stories

A user story is a method used in agile to describe functionality in a non-technical fashion, intended for the customer/user. According to M. Cohn, a user story is composed of [10]:

- A description of the story for planning
- Details to expand out the details of the story
- An acceptance criterion to determine when a story is complete

User stories should be written by the customer instead of the developers. This is to ensure that stories are written using the language of the business to avoid technical jargon. Customers also write it as they tend to have the best perspective for what the behaviour of the product will be. In 2001, W. Wake suggests the acronym *INVEST*, which are six attributes that make user stories of high quality [11]:

- **Independent** – user stories must not be dependent on one another. This means that a user story should never rely on another story to be completed. This is to reduce issues with prioritisation. It also simplifies the estimation process.
- **Negotiable** – stories must not be written as contracts or concrete requirements. The details of a story should be negotiated as part of the conversation with the customer, while the story itself should always be a simple description of a feature/function. If important details must be included, it should be added as an annotation or note attached to the user story.



- **Valuable** – stories must provide some value to either the user or purchaser/customer. The difference between a user and purchaser is, the user uses the software while the purchaser/customer owns the software. To ensure it is valuable, its best to let the purchaser/customer write it.
- **Estimable** – developers must be able to assign an estimated value to a story in terms of its size and the time it will require for it to be completed. This is very important for developers as it allows them to plan out and allocate the pace at which the project is completed. Stories may not be estimable due to a lack of domain or technical knowledge by the developers, or if the story is too large [10].
- **Small** – there's a balance to be struck over the size of stories. If it's too big or small, it's difficult to use in planning. The right size is dependent on the team's capabilities. When an entry can be broken down into multiple stories, they are called epics. Epics must be broken down into individual user stories.
- **Testable** – stories should be able to be tested to determine whether the story was completed or not. For example, "a user shouldn't have to wait long" is not testable as the time is vague whereas "a user shouldn't have to wait more than 5 seconds" is more testable.

At the DVLA, user stories are written with the format: **Given** I am an [actor], **When** I [do something], **Then** [something happens]. It covers three aspects: who I am, what I want to do, what I want to achieve. Take this user story as an example, which was taken from the DVLA agile documentation titled 'Ways of Working':

Given I am a customer using a service
When I have entered my payment details
And I have submitted the payment
And the response time is under one second
Then the "Payment Success" response is returned

This user story covers the description of the story, the details, and the acceptance criterion. It also complies with INVEST, as it is independent of other stories, can be negotiated, provides value to a user/customer, can be estimated, isn't too large or small, and it is testable by the response time.

In Jira, issues can be added to the backlog. There are four types of issues: Story, Task, Bug, and Spike as seen in Figure 4.

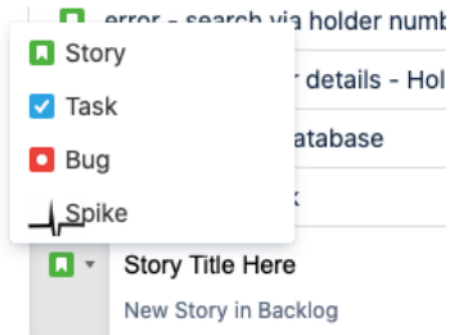


Figure 4: Screenshot of a new user story being added to the backlog.

Notice that the user story isn't included in the title itself, but rather in the issue description as seen in Figure 5.

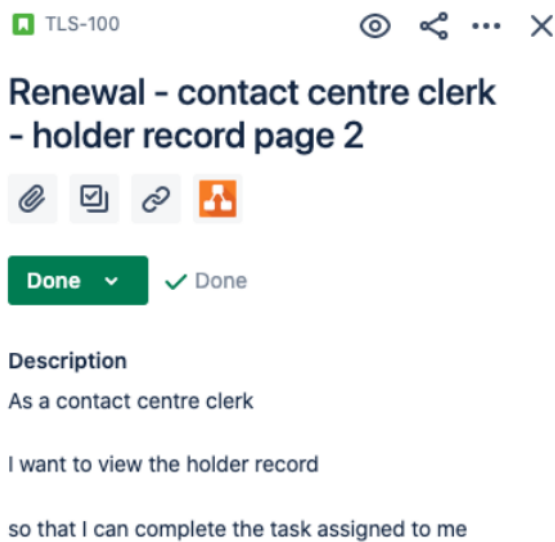


Figure 5: Screenshot of a user story description.

The tasks required for this user story can be added as subtasks, which are given a unique identifier and an assignee as seen in Figure 6.



Subtasks ... +

85% Done

FLS-141	Update enable rene...	⬆	LR	DONE
FLS-142	populate record wit...	⬆	LR	DONE
FLS-159	1. Setup controller/...	⬆	LR	DONE
FLS-160	2. Putting in the sty...	⬆	LR	DONE
FLS-161	3. Set the text	⬆	NS	DONE
FLS-162	4. Hyperlinks	⬆	NS	DONE
TLS-163	5. Manual testing	⬆	NS	TO DO

Figure 6: Screenshot of the user story's subtasks.

This example is a fairly simple story, but it can get much more complex as seen in Figure 7.

Implement Print Hub Support Tooling



Open ▾

Description

As a Printhub support developer
I want to be able to carry out support actions that
correlates to my privilege levels
So that I am able to support the application

Figure 7: Screenshot of a more complex user story in a different project.



4.1.2 Estimation

As defined in the previous section, estimation is predicting the time required for a user story to be complete. According to M. Cohn, the best approach to estimation is that it can [10]:

- Change minds based on new information
- Work with epics and stories
- Not be time-consuming
- Be valuable in giving information about the progress
- Be able to handle imprecision
- Releases can be planned

During my time working on the project at the DVLA, my team decided to base estimation off of hours **for the group total**. This was because we largely wanted to work on tasks as a group, a method known as mob programming. There are 6 team members, 6 working days (excluding the final day for review), and 6 working hours a day. This gave us 216 total hours to work with per sprint. We would estimate each user story as individuals and multiply it by 6 as there were 6 members. Estimated hours would then be subtracted from the total sprint hours until there were no more hours left, which meant the selected stories can be added to the sprint backlog. We achieved this by having each member type some hours in a group chat at the same time as to avoid influencing each other's predictions. A discussion would then take place until we decided on a number.

An example of an estimated user story can be seen in Figure 8. This user story involved creating a Redis database which none of us had done before, so we overestimated to allow room for error.

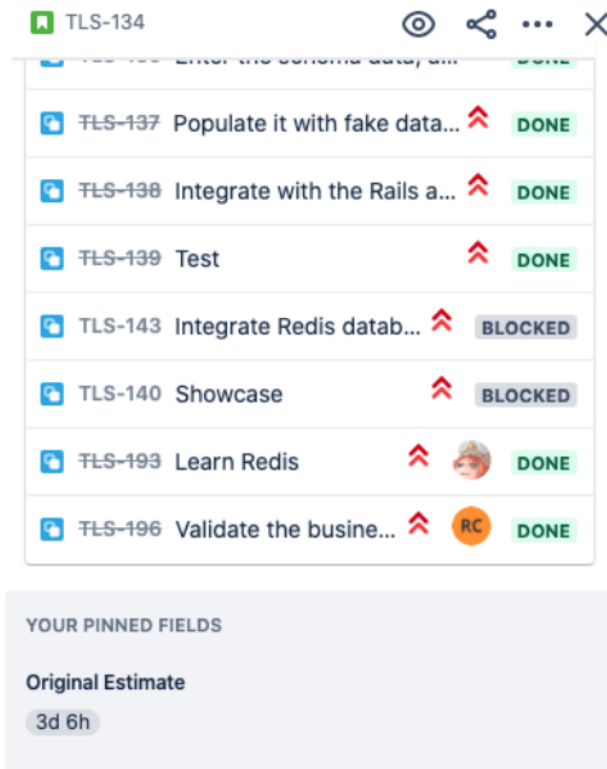


Figure 8: Screenshot of a user story with the estimate shown.

Notice that the estimate says 3 days and 6 hours. This is because, in the project setting, 1 day is defined as the number of working hours which is 6. The actual number of hours for this story was 24 hours.

4.1.3 Product Backlogs

As shown in the user story section, stories, tasks, bugs, and spikes can be added to the product backlog. Figure 9 shows the entire backlog that remained after the two sprints.



Projects / Trade Licensing Service / TLS board

Backlog

Search: [] [HB] [LR] [NS] [RB] [RC] +3 Only My Issues Recently Updated

VERSIONS

Backlog 16 issues Create sprint ...

Progress enquiry - contact centre clerk - record history - first application	TLS-107	3
Progress enquiry - contact centre clerk - search via holder details - page 1	TLS-104	-
Progress enquiry - holder record - trade licences - page 2	TLS-105	-
Search by holder details - page 1	TLS-108	-
Maintenance page - page 1	TLS-117	-
General maintenance page - page 2	TLS-118	-
No holder record history page 3	TLS-119	-
Search by plate number - page 1	TLS-110	-
search by plate number - trade licence record - business details page	TLS-111	-
error - search via holder details error -page 1	TLS-112	-
Payment errors - payment not authorised - page 1	TLS-114	-
payment error - payment failed - page 2	TLS-115	-
Payment errors - domain no response page 3	TLS-116	-
error - search via holder number error page 2	TLS-113	-
Search by holder details - Holder record - trade licences page 2	TLS-109	-
Create a Redis database	TLS-134	-
Deploy Our Work	TLS-145	-

Figure 9: Screenshot of the product backlog.

4.1.4 Sprint Backlogs

Sprints at the DVLA are usually 2 weeks long, however, this project was 2 sets of 7-day sprints. Once stories have been estimated within the constraint of the total working hours in a sprint, they are added to the sprint backlog by dragging and dropping as seen in Figure 10.

Backlog

Search: [] [HB] [LR] [NS] [RB] [RC] +3 Only My Issues Recently Updated

VERSIONS

Hyperion Squad Sprint 2 8 issues Linked pages 0 ...

Implement internal happy path and integrate database.
25/Nov/20 3:30 PM • 03/Dec/20 3:30 PM

EPICS

Renewal - contact centre clerk - Search page 1	TLS-99	8
Renewal - contact centre clerk - holder record page 2	TLS-100	5
renewal - contact centre clerk - transaction type - page 3	TLS-101	3
Renewal - contact centre clerk - Enter payment details page 4	TLS-102	5
Creating the internal repo	TLS-107	-
Renewal - contact centre clerk - Holder record - trade licences success page 5	TLS-103	3
Progress enquiry - Holder record - record history - page 3	TLS-106	-
Progress enquiry - contact centre clerk - record history - first application	TLS-107	3

Figure 10: Screenshot of Hyperion squad's second sprint backlog.



Once the sprint has started, a sprint goal is added as seen in Figure 10, under the sprint title.

4.1.5 Collaboration

To collaborate on the project, each issue is given its git branch using Jira integration, as seen in Figure 11 and 12.

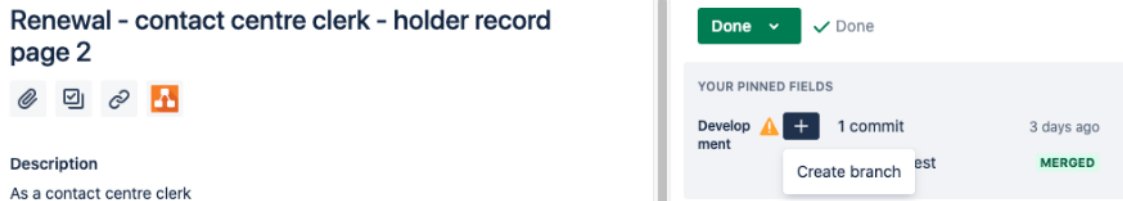


Figure 11: Screenshot of creating a branch for a user story on Jira.

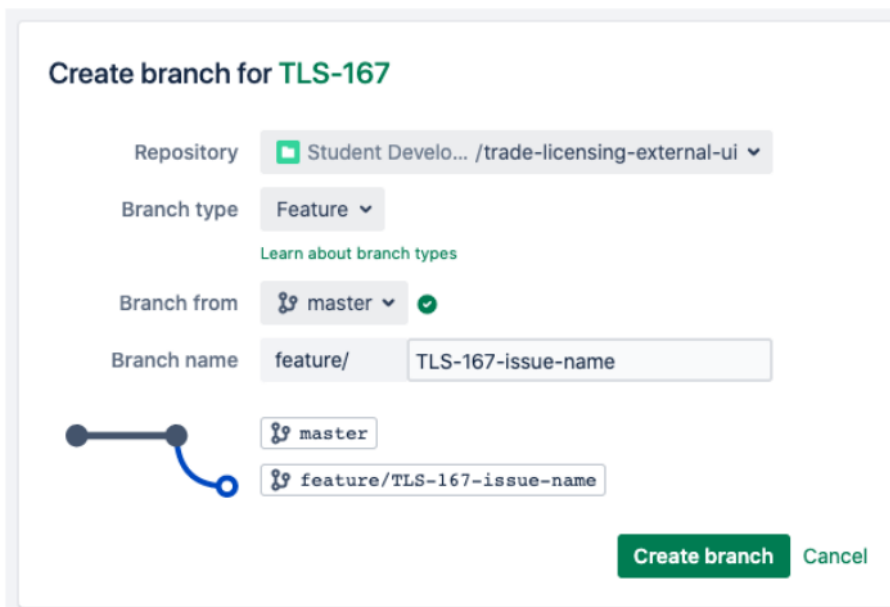


Figure 12: Screenshot of branch creation for issue on Bitbucket.

Once work has been completed in the branch, a pull request from the development branch into the master branch can be made, as seen in Figure 13.



Create pull request

Collaborate on code by choosing teammates to review your changes to a branch.

Source

Student Dev... /trade-licensi... feature/TLS-61-populate-a-database ✓

Adam Howells authored 7ca50f981a3 MERGE 18 Nov 2020

Destination

Student Dev... /trade-licensi... master

Rokas Rudys authored 3c87be3b567 MERGE 2 days ago

Continue

Figure 13: Creating a pull request in Bitbucket.

In our repository, 2 reviewers are required to approve a pull request for it to be merged into the master branch. Figure 14 shows a pull request which was approved and merged to the master branch.



redis-copy-from-internal → master **MERGED**

Redis copy from internal

[Overview](#) [Diff](#) [Commits](#) [Builds](#)

Rokas Rudys created a pull request 2 days ago

- remove methods associated with redis
- rubocop

Activity

- What do you want to say?
- Rokas Rudys** **MERGED** redis-copy-from-internal to master in commit 3c87be3b567 2 days ago
- Adam Howells** marked the pull request as **APPROVED** 2 days ago
- Adam Howells** joined as a reviewer 2 days ago
- Liam Roberts** marked the pull request as **APPROVED** 2 days ago
- Nathan Sathanandan** marked the pull request as **APPROVED** 2 days ago
- Rokas Rudys** **OPENED** the pull request 2 days ago

Figure 14: Screenshot of a pull request in Bitbucket, showing the request history.

4.2 Continuous Integration and Testing Processes

Continuous Integration (CI) is a method used to automate the building and testing of software when triggered by a git commit. M. Fowler defines CI as "a software development practise where members of a team integrate their work frequently [...] an automated build (including test) [is used] to detect integration errors as quickly as possible" [12]. This causes integration problems to be significantly reduced, According to Yang et al., the characterisations of CI include "continually changing test cases, limited execution time, and fast feedback" [13]. As developers go off to do their own thing, it's useful to have a shared version control repository where they can merge their changes frequently. Every time this occurs, the automated build and test are triggered to ensure that the master branch is valid. It's important to commit changes regularly, as it is much harder to resolve conflicts if the master and development branch diverge too much. This process can be seen in Figure 15.

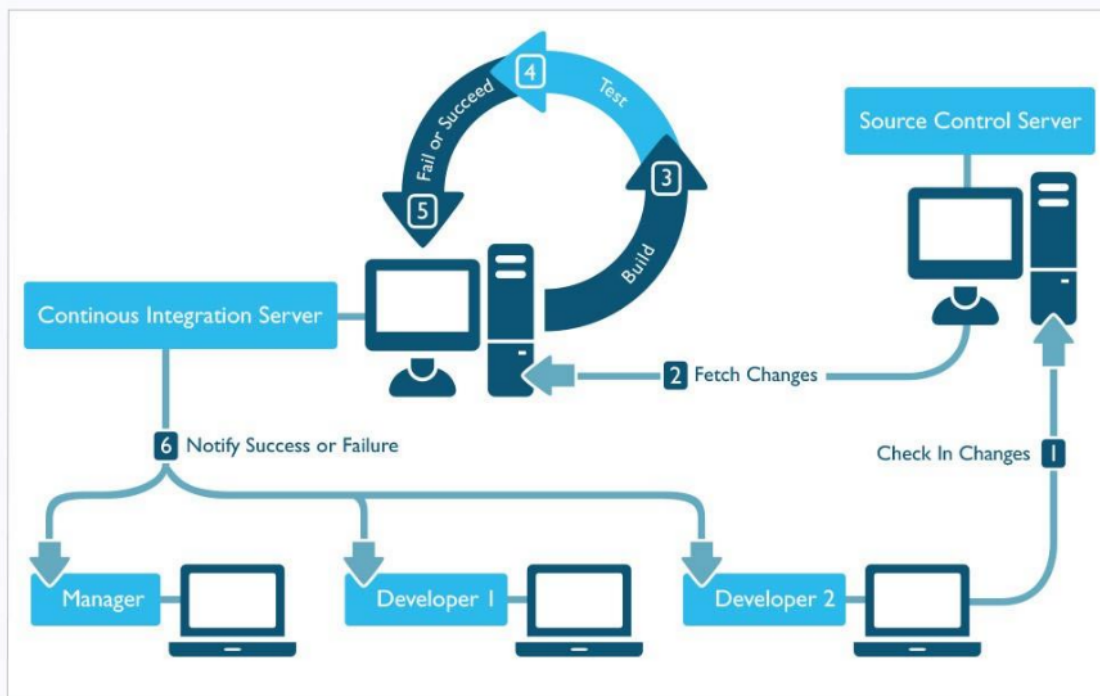


Figure 15: Diagram of Continuous Integration in DevOps. Source: <https://insights.sei.cmu.edu/devops/2015/01/continuous-integration-in-devops-1.html>.

Continuous Delivery (CD) is a discipline which allows for software to be released to production at will [14]. It is similar to CI in the sense that it automates the build and test, but it has the extra step of deployment. This is useful as the development and production builds are slightly different such as requiring different configuration files, data sources, and various optimisations. The benefit of CD is to gain fast feedback from the production to quickly fix any issues or defects, something that might be expensive otherwise [14]. Figure 16 shows the relationship between CI and CD.

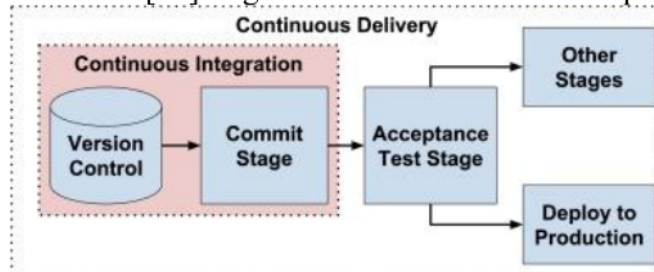


Figure 16: Diagram of CI and CD. Source: <https://www.sciencedirect.com/science/article/pii/S0950584916302324>.

Adam Howells



CI/CD uses a concept called pipelines. Pipelines are a deployable unit path. An example of a pipeline might be to build, run unit tests, security tests, and then deploy to a destination. The DVLA uses drone.io as their CI software [15], which features a powerful cloud-native pipeline engine. Drone.io can be integrated into Bitbucket where it will build the software on commit. Figures 17 and 18 show this in action.

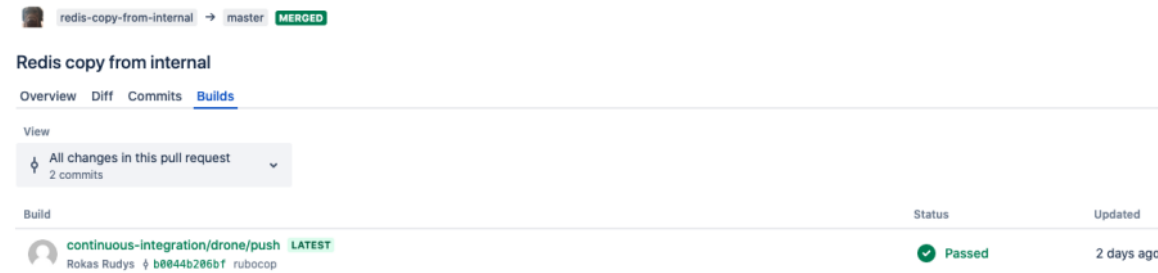


Figure 17: Screenshot of a pull request's build in Bitbucket.

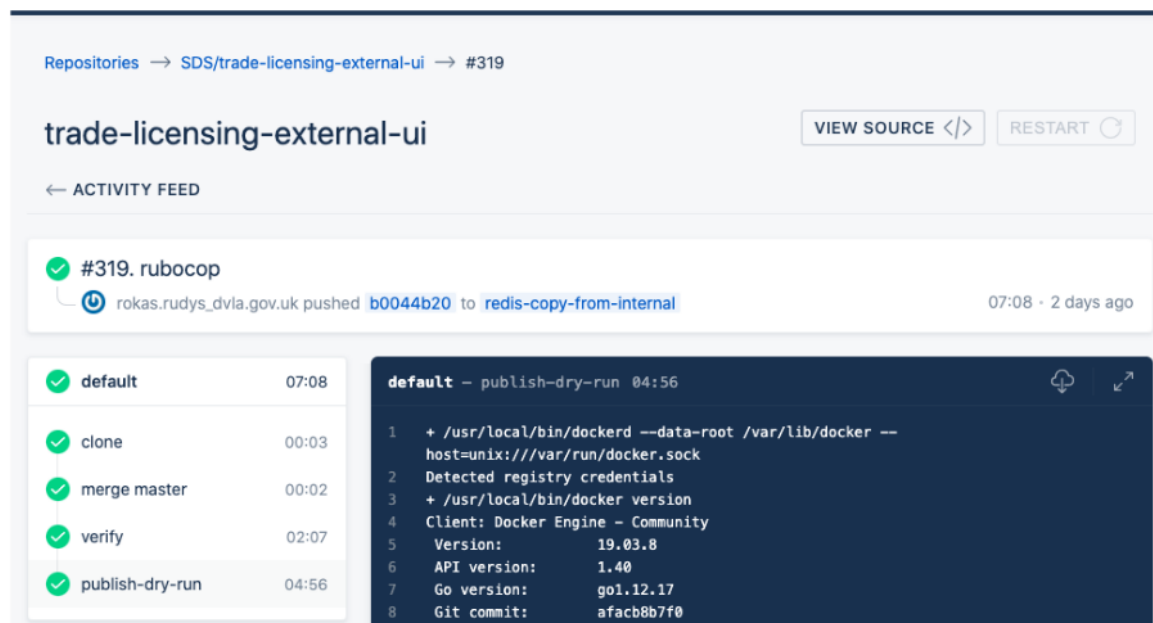


Figure 18: Screenshot of a successful CI build in Drone.io.

The pipelines mentioned earlier can be seen on the left-hand corner in Figure 18. As this is a merge, it first clones the development branch, merges it with the master branch, verifies it (checking code formatting etc.) and then builds and publishes the docker container. The console can be viewed to identify errors for when it fails at any stage.



5 Bibliography

- [1] F. Hayat, A. U. Rehman, K. S. Arif, K. Wahab, and M. Abbas, 'The Influence of Agile Methodology (Scrum) on Software Project Management', in *2019 20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, Jul. 2019, pp. 145–149, doi: 10.1109/SNPD.2019.8935813.
- [2] M. Fowler and J. Highsmith, 'The agile manifesto', *Softw. Dev.*, vol. 9, no. 8, pp. 28–35, 2001.
- [3] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, 'Agile Software Development Methods: Review and Analysis', *ArXiv170908439 Cs*, Sep. 2017, Accessed: Nov. 22, 2020. [Online]. Available: <http://arxiv.org/abs/1709.08439>.
- [4] K. Schwaber and J. Sutherland, 'The scrum guide', *Scrum Alliance*, vol. 21, p. 19, 2011.
- [5] K. Schwaber and M. Beedle, *Agile Software Development with Scrum*. Prentice Hall, 2002.
- [6] A. Srivastava, S. Bhardwaj, and S. Saraswat, 'SCRUM model for agile methodology', in *2017 International Conference on Computing, Communication and Automation (ICCCA)*, May 2017, pp. 864–869, doi: 10.1109/CCAA.2017.8229928.
- [7] Atlassian, 'Development and Collaboration Software Company', *Atlassian*. <https://www.atlassian.com/company> (accessed Dec. 04, 2020).
- [8] Atlassian, 'Jira | Issue & Project Tracking Software', *Atlassian*. <https://www.atlassian.com/software/jira> (accessed Dec. 04, 2020).
- [9] Atlassian, 'Atlassian Bitbucket Git Code Management Tool for Teams', *Atlassian*. <https://bitbucket.org> (accessed Dec. 04, 2020).
- [10] M. Cohn, *User Stories Applied: For Agile Software Development*, 1 edition. Boston: Addison-Wesley Professional, 2004.
- [11] W. C. Wake, *Extreme programming explored*. Boston : Addison Wesley, 2002.
- [12] M. Fowler and M. Foemmel, *Continuous integration*. 2006.
- [13] Y. Yang, Z. Li, L. He, and R. Zhao, 'A systematic study of reward for reinforcement learning based continuous integration testing', *J. Syst. Softw.*, vol. 170, p. 110787, Dec. 2020, doi: 10.1016/j.jss.2020.110787.
- [14] E. Laukkanen, J. Itkonen, and C. Lassenius, 'Problems, causes and solutions when adopting continuous delivery—A systematic literature review', *Inf. Softw. Technol.*, vol. 82, pp. 55–79, Feb. 2017, doi: 10.1016/j.infsof.2016.10.001.
- [15] 'Drone CI – Automate Software Testing and Delivery'. <https://www.drone.io/> (accessed Dec. 04, 2020).

FINAL GRADE

81 / 100

GENERAL COMMENTS

Instructor

Agile Process Evaluation (40): 35

Examples (30): 25

Testing (30): 21

Discussion:

Excellent discussion of the agile process applied in DVLA, with a detailed breakdown.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21
