Robert Hill

Student number: 189047532

Student ID: rdh36

Word count: 13908

Project Genesis

A Software Development Fantasy

Software Development Fantasy

Dissertation

Robert Hill

Student number: 189047532

Student ID: rdh36

1 ABSTRACT 4

2 THE PROBLEM 5

2.1 AGILE 5
2.2 QUALITY ASSURANCE 5
2.3 SOFTWARE ENGINEERING 5

3 PROJECT AIMS AND OBJECTIVES 7

3.1 REALISM AND GAMIFICATION 7

4 LITERATURE REVIEW 8

4.1 GAMIFICATION 8
4.2 AGILE PRACTICES 8
4.3 QUALITY ASSURANCE PRACTICES 9
4.4 SOFTWARE ENGINEERING PRACTICES 9

5 SPECIFICATION 10

5.1 OVERVIEW 10
5.2 SIMULATION 10
5.3 GAMEPLAY 11
5.4 PLATFORM REQUIREMENTS 13

6 IMPLEMENTATION 14

6.1 SOFTWARE DESIGN 14
6.2 MODELLING AND SIMULATION 22
6.3 SIMULATION: THE SPRINT 27
6.3.6 PLAYER LEARNING 29

7 EVALUATION 31

7.1 USER ASSESSMENT METHODOLOGY 31
7.2 SURVEY QUESTIONS 31
7.3 LIMITATIONS 33
7.4 SURVEY RESULTS 34

8 ANALYSIS 43

8.1 IMPLEMENTATION 43
8.2 GAME RESULTS 44
8.3 PLAYER SURVEY 46
8.4 THE GAME AS A LEARNING TOOL 47

9 FUTURE WORK 49

9.1 INTENDED AUDIENCE 49
9.2 PLAYER LEARNING 49
9.3 DIVERSITY 49
9.4 PLAYER KNOWLEDGE 49

Robert Hill

Student number: 189047532

Student ID: rdh36

9.5 Additional Discovery Points 49
9.6 Randomness 50
9.7 Resignations, redundancies & happiness 50
9.8 Agile 50
9.9 Estimation, Velocity and Sprint Planning 50
9.10 Agile Mindset 50
9.11 Customer Satisfaction 50
9.12 Player observation of "hidden" areas of the model 51

10 CONCLUSION 52

11 REFERENCES 53

Software Development Fantasy

Dissertation

Robert Hill

Student number: 189047532

Student ID: rdh36

1 Abstract

Software development is complex, is easy to get wrong, and it can be difficult to identify and understand the causes of failures without appropriate knowledge and (sometimes years of) experience. Successful software projects and the engineering teams that produce them are comparatively rare. Most software professionals learn software development principles and practices by absorbing the practices used in their places of work, often building up an impressive amount of misunderstanding and misuse along the way.

Agile practices are poorly understood and badly applied. Quality Assurance practices are patchy. Software engineering practices can be missing from many software engineers' arsenals. Few of these practices are taught well or at all in Computer Science degree modules.

Yet, there are well-defined practices within the industry that if understood and followed, can lead to greater quality, stability and reliability in software development projects.

This research project aims to highlight and teach the relevance and importance of 23 software development principles or practices, in the three broad categories of Agile, quality assurance and software engineering. The pedagogical vehicle is a browser-based game, modelled on Game Dev Tycoon and similar "game dev" games. The game aims to teach the player not only about the 23 software development principles or practices directly, but also to teach them through experience what impact these principle and practices have on the success or failure of a software project.

The project includes analysis of the players' experience and learning via a self-reported survey.

Robert Hill

Student number: 189047532

Student ID: rdh36

2 The Problem

Software development is complex, with many interconnected processes and practices involved. Real-world software development practices are generally not taught in formal education. Best practices evolve and are popularised and discussed online, in books, at conferences and in the workplace. Practices are usually learned through experience in the workplace and may easily be misunderstood or misapplied.

Software development is easy to get wrong, and it can be difficult to identify and understand the causes of failures without appropriate knowledge and (sometimes years of) experience.

If you break the software development process down into 3 broad areas, the problems associated with each area can be more easily understood:

- Agile practices
- Quality Assurance practices
- Software Engineering practices

2.1 Agile

Discussions about introducing agile practices into an organisation generally start from the premise that we start with a baseline of Waterfall existing at the organisation, but nowadays this is rarely the case. It's far more likely that everyone in the organisation has some knowledge and experience of Agile, but little or no training and very often the Agile practices and their functions will be misunderstood and misapplied (Javdani et al. 2015)[i].

2.2 Quality Assurance

Likewise, some QA function will likely exist within an organisation, but the quality of quality assurance practices can vary wildly from one organisation to another or even within different parts of the same organisation. In my own personal observations during a 20-year career, the perception of QA as a separate function from software engineering, carried out by different people, can have a hugely negative effect on the overall quality of processes and software deliveries.

2.3 Software Engineering

To understand what can go wrong in an organisation's software engineering practices, it is worth comparing how software engineering differs from programming as both an activity and a career. Programming is the act of writing code in a programming language to produce some software that performs some function, usually to meet some requirements. Software engineering includes programming, plus other practices, such as:

- Software design
- Unit/integration/E2E/etc. testing
- System architecture
- Automated builds/deployments

- Code review

The software engineering culture within an organisation may also include other factors which affect the quality of software engineering practices, such as:

- Tech talks / knowledge sharing
- Pair programming
- Involvement in candidate interviews
- Inclusivity
- Psychological safety
- Collaboration

Robert Hill

Student number: 189047532

Student ID: rdh36

3 Project Aims and Objectives

This software development fantasy project aims to model the software development process as a game, and educate the user about how to apply good practices in software development. As a model, the game should contain enough interacting elements to portray a realistic representation of the software development process, and the decisions and compromises that need to be made in a real-world software project. The model should also be simple enough to allow the user to see and understand how their choices impact the successes and failures of a software project in progress.

There is no single right way to do software development, but there are processes and practices that are generally better than others in certain circumstances. The game will aim to demonstrate the trade-offs that must be made to try to steer the right course through a software project, and give regular feedback to the user to help them understand the impact of their decisions.

One of the reasons for the need for an educational game of this type is the widespread misuse or misunderstanding of Agile, QA and software engineering practices. The project aims to demonstrate the value of using these practices well, and demonstrate the importance of using them together effectively.

The project aims to educate the player about specific areas of software development with which they may be unfamiliar.

Where possible, existing research has been used to determine the importance and impact of software development practices in the game. Where this is not possible, generally accepted industry best practices have been used.

3.1 Realism and Gamification

Accurately modelling the software development process well enough to simulate all the moving parts of a software project is difficult. Understanding the relative weights and impacts of the inputs takes trial and error, with little published research to rely on. In general, the model and simulation have been balanced to “seem” real in the context of the game because more than anything else this game is a teaching tool and realism is less important that “perceived” realism and playability.

Backing up the relative benefits of different practices with existing research has been difficult in some areas. Where this was not possible, widely accepted best practices will be used instead.

Robert Hill

Student number: 189047532

Student ID: rdh36

4 Literature Review

In reviewing existing literature, I have focused on research into gamification, and research into software development best practice in the three categories previously discussed: Agile, Quality Assurance and Software Engineering.

4.1 Gamification

Code Defenders (Rojas et al. 2017)[ii] is an example of Gamification taking advantage of the players' competitive natures through attacker/defender, point-based and prize reward mechanisms.

Krafft et al. (2020)[iii] discuss the use of a block-based programming language, Scratch, to teach adults programming concepts, with positive results. Scratch includes a visual, drag-and-drop authoring tool that conveys programming constructs like loops and conditionals as if they were physical objects that can be manipulated on screen with a mouse/keyboard.

Vos et al. (2019)[iv] discuss the current progress of the IMPRESS project on Gamification, covering quizzes, gamification of formal specification creation, teaching through competition, and AI.

Hamari et al. (2014)[v] conducted a review of existing research into gamification. They discuss what gamification is conceptually, discuss both positive and negative outcomes of the efficacy of gamification and discuss whether benefits of gamification may be long-term .

4.2 Agile Practices

SCRUMIA (Wangenheim et al. 2013)[vi] discusses a paper-and-pencil game used to teach Scrum to Project Management students. It covers one Sprint and takes a hands-on, visual and interactive approach to maximise the learning outcomes.

A decade of agile methodologies: Towards explaining agile software development (Dingsøyr et al., 2012)[vii], provided some insight into the early years of research into Agile. Initially focussing on Agile adoption and the concept of Agility, research later began to shift to the evaluation of Agile practices.

Rauf and AlGhafees (2015)[viii] reported a generally high acceptance of the benefits of a broad range Agile practices among managers, developers and other staff, even in cases where some of the practices are little-used.

Very little work has been done on Agile retrospectives. In a longitudinal case study, Lehtinen et al. (2017)[ix] showed some issues with the practice of Agile retrospectives within a single organisation, particularly around repetition and a lack of actioned outcomes.

Other work (Matthies et al., 2019)[x] has demonstrated differences in the value of retrospectives depending on the practices used, and broadly back up the value of structured practices promoted by the Scrum community.

These two studies on Agile retrospectives highlight an issue with Agile that has been apparent in my own experience as a software engineer and scrum master. Agile done badly may be worse than no Agile at all.

In one study on teaching Scrum practices (M. Paasivaara, 2021)[xi], M. Paasivaara reported on the effectiveness of using professional Agile coaches to teach and guide Scrum Master in somewhat realistic settings. They also reported on the suitability of women and people with non-technical backgrounds for the role of Scrum Master. The study also comments on how unprepared most Scrum Masters are for the role, even after training and certification, and how uncommon Agile coaching is in the industry.

In previous work (Paasivaara and Lassenius, 2016)[xii], the importance of training, coaching and mindset was highlighted in the success of large-scale Agile transformations.

In general, it had been difficult to identify research into the quality of Agile specific practices used in the industry, or the consequences of doing Agile well or poorly.

4.3 Quality Assurance Practices

In *When, how, and why developers (do not) test in their IDEs*[xiii] (Beller et al., 2015), it is demonstrated that in a sample of Java developers, unit testing is very often not carried out, test driven development (TDD) is rare and the time spent writing tests is significantly less than time spent writing code. In later work (Beller et al. 2019)[xiv] the same results were demonstrated for C# developers.

Wiklund et al. (2017)[xv] discuss the challenges of test automation covered by existing research, helping to illuminate best practice in the process.

4.4 Software Engineering Practices

Rauf and AlGhafees (2015) also cover software engineering practices originating in the Xtreme Programming movement that are widely recommended as part of Agile development, such as refactoring, pair programming, collective code ownership and test-driven development (TDD). They reported that these are generally thought of as good practice by developers, even when the practices are not followed.

Dybå et al. (2007)[xvi] found clear benefits of pair programming in terms of quality, and lesser but measurable benefits in terms of overall duration and effort.

I have been unable to find relevant literature on the general benefits of DevOps, Continuous Integration and Continuous Deployment.

Robert Hill

Student number: 189047532

Student ID: rdh36

5 Specification

5.1 Overview

The software development fantasy game “Project Genesis” puts the player in the role of a CTO (Chief Technology Officer) at a small but growing software development company, in charge of a new project, “Project Genesis”.

As CTO, the player must build a team of talented software professionals and guide them to success, both in the delivery of a project to a customer, but also in improving the quality of software development practices within the organisation.

Throughout the game there are opportunities for the team to discover new practices and deepen their knowledge of Agile, quality assurance and software engineering.

The goal will be to deliver a project on time and within budget while meeting your customer’s changing expectations. The player will need to do this while starting with a somewhat dysfunctional organisation that already uses Agile practices, but lacks some key team members, such as testers, a scrum master and a product owner.

Much of the organisation state will be hidden from the player, and they will need to infer how things are going by indirect means, such as the number of bugs, the customer’s feedback or how efficiently the team complete work.

Along the way, the player will hopefully learn more about important software development practices in three categories:

- Agile
- Quality Assurance
- Software Engineering

Each of the practices is widely known in the industry, and important to the success and stability of modern software systems. The player should also start to learn how using these practices can positively impact the software development process, and which skills or attributes they develop and relate to in team members.

5.2 Simulation

The game should simulate the software development process in the following areas:

- Agile
    - Scrum Events
    - Team self-organisation
    - Team communication
    - Team collaboration
    - Responding to changing priorities

    - Usage of well-known Agile best practices (such as 3 Amigo sessions)
    - Continuous improvement
- Quality Assurance
    - Test design
    - Automated testing
    - QA in all stages of development (or testing at every point in the process, rather than the end)
- Software Engineering
    - Software design
    - Pair programming
    - Refactoring
    - Code review
    - CI/CD
    - Tech talks
- Firefighting

5.3 Gameplay

5.3.1 Starting Out

The game begins with the player accepting the role of CTO at one of three companies.

Before game rounds commence, the player will have the opportunity to perform actions to understand the current state of the company, such as:

- The team members' skills and motivations.
- The quality of current processes and practices.
- The customer's needs.

These actions are optional and can be repeated later, during the game.

Based on their knowledge of the current state of the company, the player can also choose to spend time improving the team's skills and practices by holding workshops on software development topics, such as:

- Tech talks.
- Software design.
- Unit tests.
- Pair programming.
- CI/CD pipelines.
- Code review.

Though they do so with some risk of delaying the delivery of features and a dissatisfied customer. The number of topics available increases as the game progresses, representing areas of discovered knowledge.

At the beginning of the game, the player must take on some of the responsibilities of a scrum master and product owner, until they are able to hire people into these positions.

5.3.2 The Sprint

Each game round is represented by a Sprint, a time-boxed period of work in which the team attempt to deliver their goal, in the form of the next product increment. Initially the player must manage the Sprint Events (Sprint Planning, Daily Scrum, Sprint Review and Sprint Retrospective), and handle any unexpected situations.

The player will need to pay attention to the customer and their changing priorities, helping the team to deliver value to the customer. At the same time, other priorities will need to be balanced, such as reducing bugs, maintaining quality, and firefighting.

The success or failure of the Sprint will depend on many factors. The amount of successful work done by the team depends on their skill, experience, and level of collaboration. The number of bugs created by the team will depend on their skill, quality mindset and level of collaboration. The level of customer satisfaction will depend on whether the team prioritised and delivered what the customer asked of them.

At any point during the Sprint, unexpected events can happen, forcing the player and team to choose between meeting the customer's needs and firefighting. A good CTO will find ways to stabilise the organisation and reduce the need to firefight.

As soon as one Sprint ends, the next is prepared. The player can interact with other areas of the game between sprints.

5.3.3 Outside the Sprint

The CTO also needs to take care of other matters while the Sprints are ongoing. For example:

- Keeping abreast of the customer's changing needs.
- Meeting the recruiter and hiring new staff to fill existing and new vacancies.
- Discovering more about the team's skills and knowledge.
- Holding workshops to spread knowledge of software development practices around the team.
- Hiring consultants to improve practices.

5.3.4 The importance of the 5 roles

There are 4 roles available to hire:

- Software Engineer

    - Responsible for doing the bulk of the work on the project.
- QA Engineer
    - Responsible for doing work on the project.
    - For the team to "discover" QA practices in Retrospectives, there needs to be at least 1 QA Engineer on the team.
    - Each QA Engineers boosts the team's quality mindset characteristic by 10% (up to a maximum of 1).
- Scrum Master
    - For the team to "discover" Agile practices in Retrospectives, there needs to be a Scrum Master on the team.
    - The team can only have 1 Scrum Master.
- Product Owner
    - If the team have a Product Owner, the Product Owner moves the 2 features prioritised by the customer to the top of the backlog before Sprint Planning begins.
    - The team can only have 1 Product Owner.

There is 1 consultant that can be hired for a short period:

- Agile Coach
    - The Agile Coach works with the team for a few days and has some probability to improve each team member's agile mindset. This also causes a distraction and limits the amount of work the team can do while the consultant is present. The team members' agile mindset is an important characteristic, because it impacts how well the team apply other improvements originating from the team's Retrospectives.

5.3.5 The (Hidden) Goal

The overt goal is to deliver a project on time and within budget while meeting the customer's expectations.

The real goal of the game is to discover, introduce and use practices that result in the streamlined, efficient delivery of working, well-designed and well-tested software.

5.4 Platform requirements

The game should be built in HTML and JavaScript to enable it to be played on and ported to any web-capable device. For the purpose of this research project, it has been built to run on Chrome Desktop only.

Robert Hill

Student number: 189047532

Student ID: rdh36

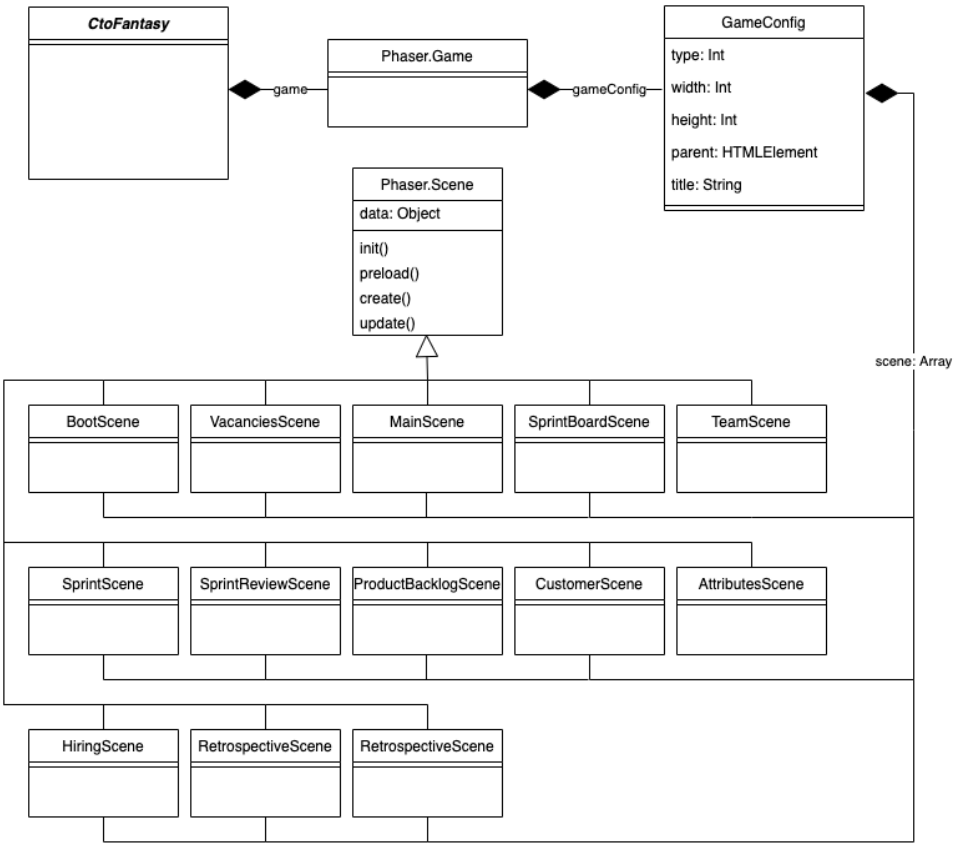
6 Implementation

The game uses Phaser, a JavaScript game programming framework that renders graphics to a web canvas or via WebGL depending on the capabilities of the browser. For this research project, the game has been built to run on the Chrome web browser on desktop only, but with the intention that this could be ported to any web-capable device, or packaged easily as a mobile app and made available on any modern phone or tablet with limited effort.

6.1 Software Design

6.1.1 Game and Game Scenes

The primary module in Phaser is the Scene. Phaser Scenes have lifecycle methods making it easy load new scenes and navigate from one scene to another. Scenes are typically used as controllers in an MVC-like pattern. The game is comprised of 13 scenes that extend Phaser's Scene class: BootScene, VacanciesScene, MainScene, SprintBoardScene, TeamScene, HiringScene, SprintScene, SprintReviewScene, ProductBacklogScene, CustomerScene, AttributesScene, RetrospectiveScene, and EndScene. The main class (CtoFantasy) creates an instance of the Phaser.Game class, passing references to all of the scene Classes (not instances of them) as part of Phaser's GameConfig. Phaser instantiates these scene classes during the game boot process.



Class diagram showing the main CtoFantasy class and Scene classes.

6.1.2 Linear State Machine

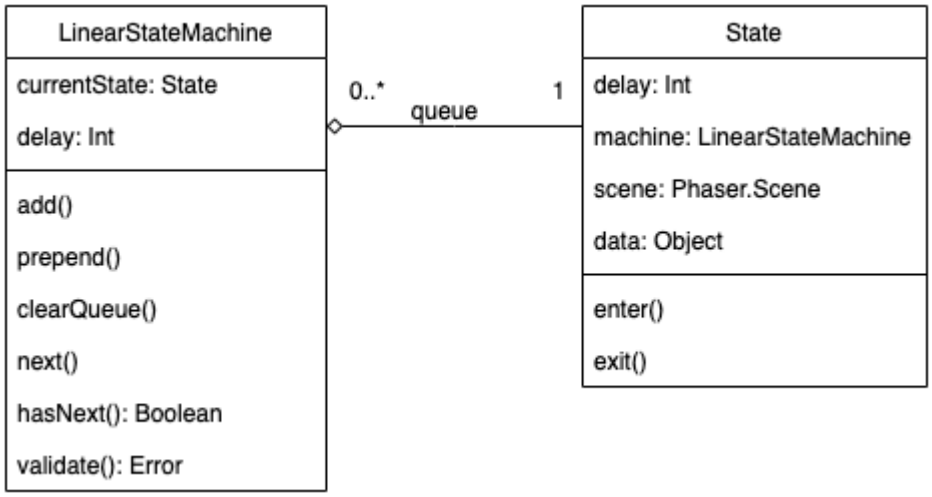
Several approaches to game state transitions were experimented with. Initially an event-based PubSub approach was used, then later a Finite State Machine (FSM). The event-based approach was a little disorganised for state handling (though retained for decoupled inter-component communication). The FSM was too restrictive for the iterative approach I have taken during development, so a simpler Linear State Machine (LSM) was created. Multiple instances of the LSM can be instantiated. Polymorphic state objects can be added to either end of a queue held within the LSM, and the LSM calls the states' lifecycle functions and handles transitions between states. The states run arbitrary code when opened and closed, within their lifecycle methods.

The LSM offers a very flexible way of queueing up a sequence of game events at the start of the game, but also being able to add any new events to the first position in the queue if they need to be resolved before the rest of the sequence continues. The game creates 1 instance of the LSM at the start of the game to hold the sequence of Sprint states, and each SprintScene creates an instance of the LSM to hold a sequence of states use within each Sprint.

Robert Hill

Student number: 189047532

Student ID: rdh36



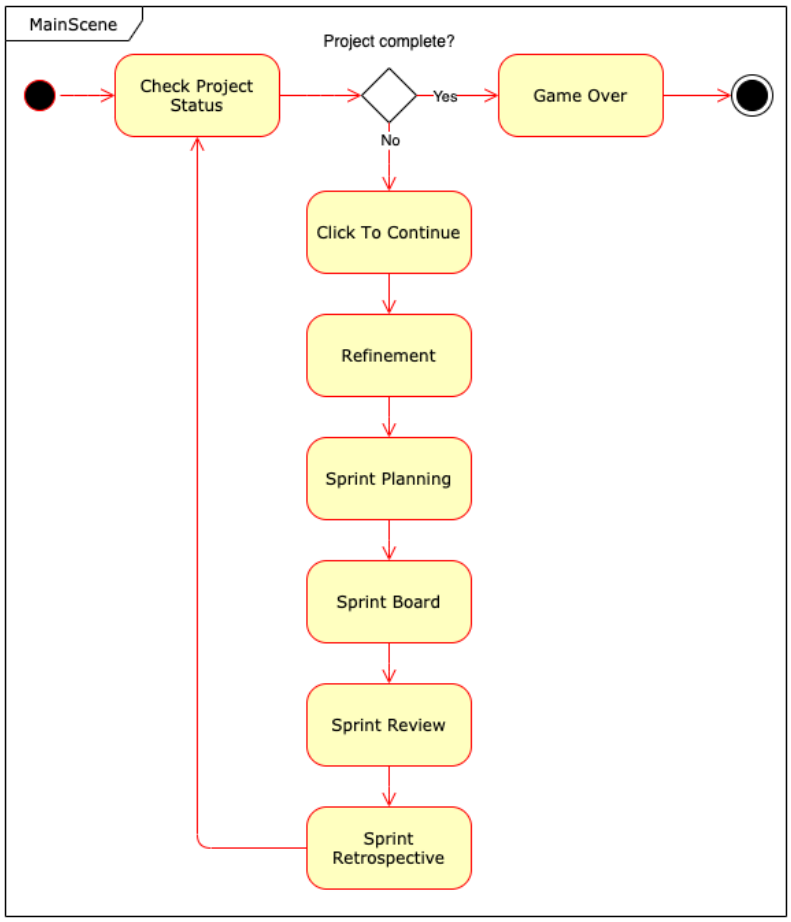
Class diagram showing the LinearStateMachine class and State superclass.

A generic NavigationState was created and a navigationStateFactory method to encapsulate wrapping a state around a Phaser Scene. This enables the LSM to navigate between Phaser scenes using the states' lifecycle methods. In other cases, specific State classes were created to enable more detailed setup and teardown around the scene transition.

Robert Hill

Student number: 189047532

Student ID: rdh36



A state diagram showing the states that the Linear State Machine cycles through until the game ends. Game end is determined by either 10 sprints being completed, or all stories in the backlog being completed (whichever occurs first).

Some of the state transitions are not visible to the player.

- ***CheckProjectStatusState*** acts as a decision node to determine if the game should end.
- ***RefinementState*** represents the team refining and estimating and user stories prior to the sprint. This happens without player involvement.

The ***ClickToContinueState*** acts as a pause mechanism between sprints to allow the player to perform other actions, such as hiring new employees, meeting with the customer, viewing and modifying the backlog, or arranging workshops.

The following states all open a corresponding Phaser scene, and include a delay in milliseconds as a property of the state to control how the Linear State Machine opens the state:

- ***SprintPlanningState*** opens the ***ProductBacklogScene*** with additional launch parameters to ensure additional text and components relating to estimation and velocity are displayed.
- ***SprintBoardState*** opens the ***SprintBoardScene***.

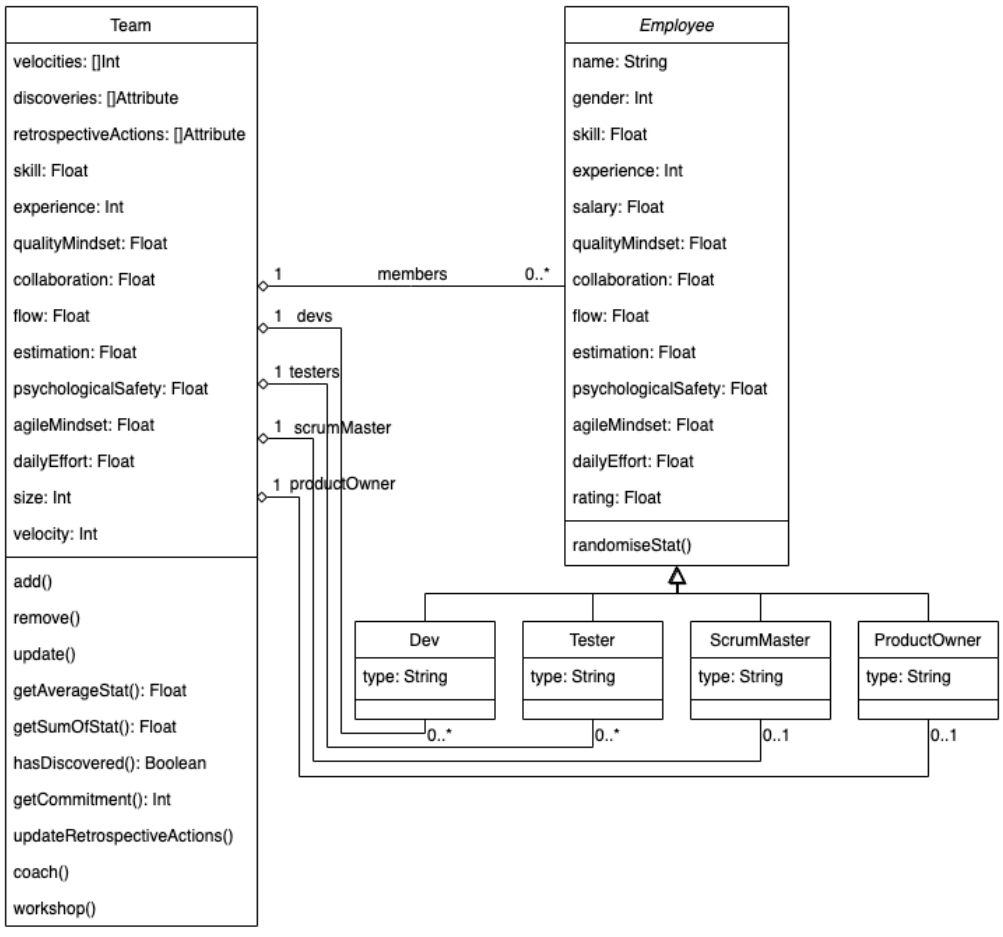
- ***SprintReviewState*** opens the ***SprintReviewScene***.
- ***SprintRetrospectiveState*** opens the ***SprintRetrospectiveScene***.
- ***GameOverState*** opens the ***GameOverScene***.

6.1.3 Data Model

The game's data model is represented by a number of classes.

6.1.3.1 The Team

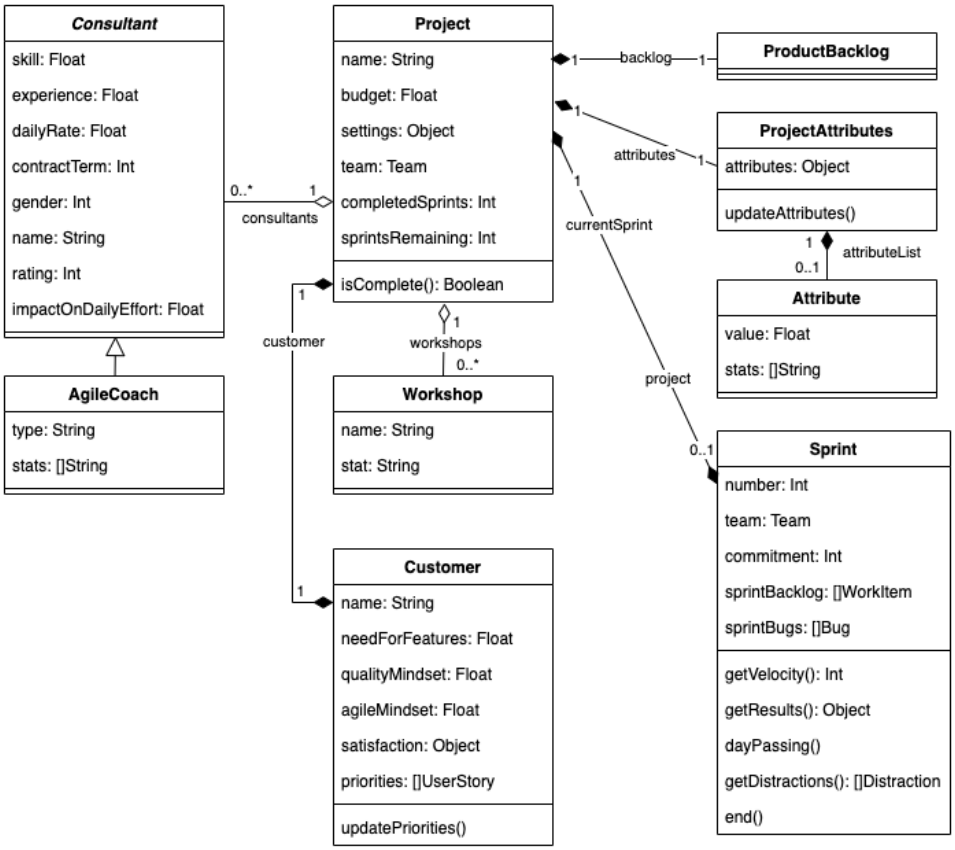
Each employee is instantiated as an existing team member or instantiated as a candidate and made available for hire via the recruiter. Employees have randomly generated attributes. The team is made up of all the hired employees. The majority of the team's attributes are derived from the average of its members. The Team class also tracks information about the team's changing velocity through the game, the discoveries the team has made and the retrospective actions the team intend to perform each sprint.



The Team

6.1.3.2 The Project

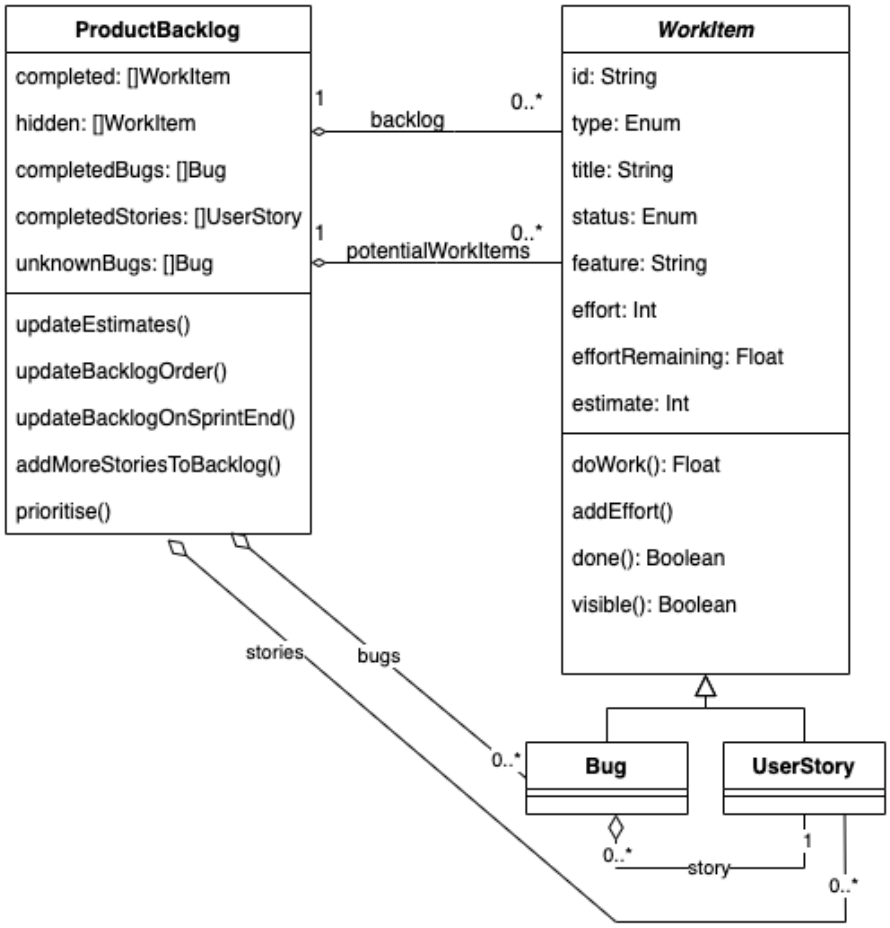
The Project class holds information about the progress and status of the project, the customer, any currently hired consultants, the current sprint, the product backlog, and the project attributes (the software development practices that the player is attempting to improve).



The Project

6.1.3.3 The Product Backlog

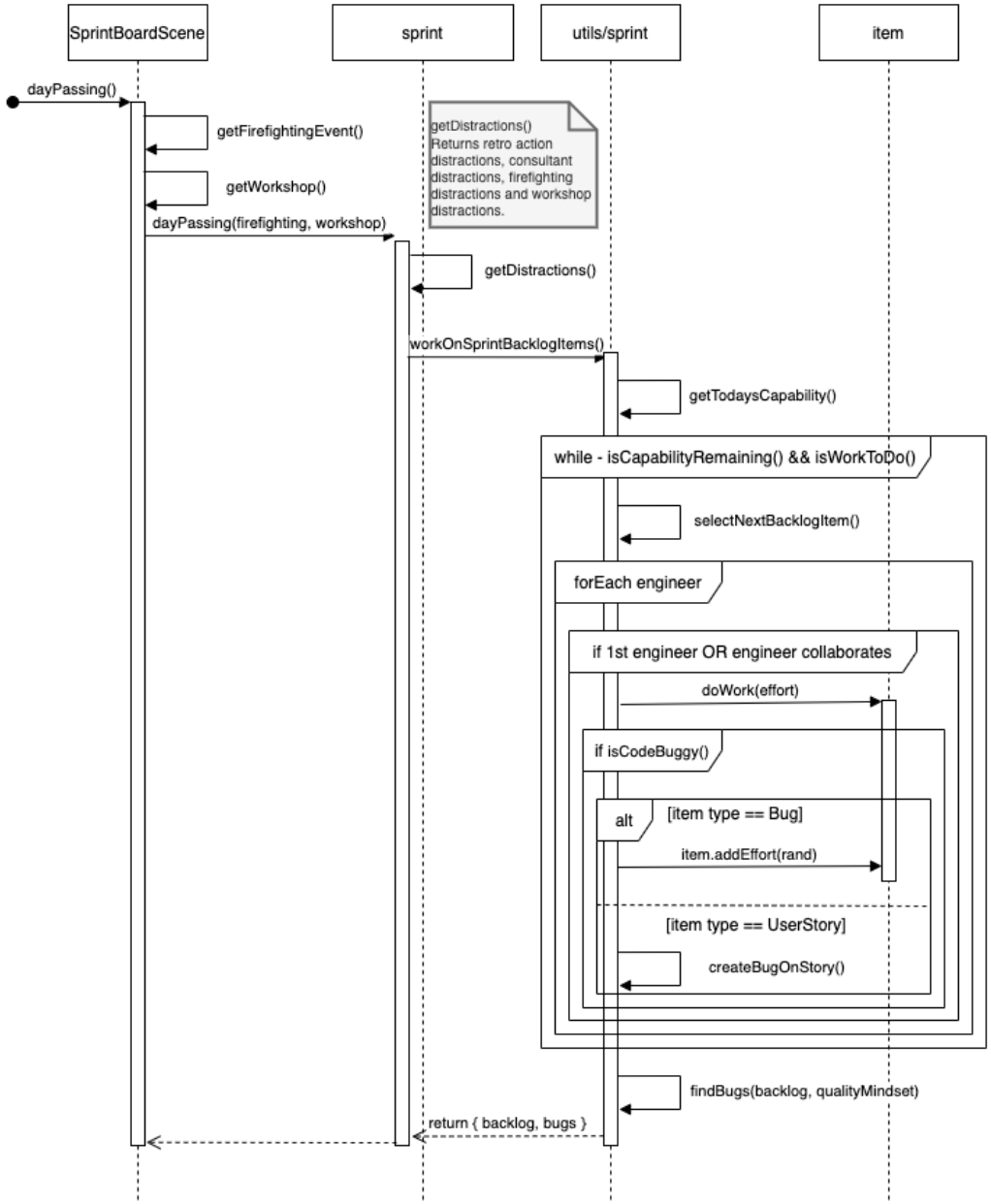
The product backlog represents all of the features required to meet the customer's needs and all of the bugs contained in the product (including those that have not been identified by the team or customer).



The Product Backlog

6.1.4 Simulating a day of work

A large proportion of the business logic in the game is executed during the Sprint, while the ***SprintBoardScene*** is active. This is the primary area of simulation in the game, where the simulation business rules are applied to the current data model and the model is updated.



The simulation of a day passing in the sprint.

Robert Hill

Student number: 189047532

Student ID: rdh36

6.2 Modelling and Simulation

In order to teach the player about software development practices via an imaginary software project, it is necessary to create a model of the software development process that is detailed enough to allow for a realistic simulation. The model is an abstraction (and simplification) of the real world, in this case for the purpose of teaching the fundamentals of good software development practice.

The core of the model is the game state, which is invisible to the player. On top of the game state are further layers of the model which the player has some visibility of.

6.2.1 Model: Game State

The game state is represented by a small number of properties:

- Project budget
- Current Sprint number
- Product backlog
- Employees
    - Skill
    - Experience
    - Gender
    - Salary
    - Quality mindset
    - Collaboration
    - Flow
    - Estimation
    - Psychological safety
    - Agile mindset
- Team
    - All team characteristics are derived from the team members (usually as an average).
- Customer
    - Priorities
    - Quality mindset
    - Satisfaction

Some of the above properties are not self-explanatory.

- *Quality mindset* represents how an employee thinks about and values quality, both in the sense of QA testing and applying a critical approach to quality in all other areas of software development. The customer's quality mindset impacts how likely they are to find hidden bugs, and how accepting they are of bugs in the Product Backlog.
- *Flow* is often used as a metric in Kanban teams to represent the efficiency of progress of work. It is used here to represent the mindset and behaviours that an individual has that promote flow,

primarily the behaviour of working collectively on items in priority order to get the highest priority work completed first.

- *Psychological safety* represents the comfort the employee has in openly speaking their mind without fear of unconstructive criticism or retribution.
- *Agile mindset* represents the employee's willingness to accept and participate in agile practices, and their knowledge of and ability to perform agile practices.

6.2.2 Model: Software Development Practices

The software development practices used in the project represent the collective knowledge and experience of the whole team in the 3 core areas. Their current values are derived from one or more of the team's characteristics (in brackets below):

- Agile
  - 3 Amigos (Agile mindset, quality mindset, collaboration)
  - Refinement (Agile mindset, estimation)
  - Daily Scrum (Agile mindset, collaboration)
  - Retrospective (Agile mindset, collaboration, psychological safety)
  - Review (Agile mindset)
  - Psychological safety (Psychological safety)
  - WIP (work in progress) limit (Agile mindset, collaboration, flow)
  - Sprint goal (Agile mindset, collaboration)
  - Continuous improvement (Agile mindset, quality mindset)
  - Customer engagement (Agile mindset)
- Quality Assurance
  - Test design (Quality mindset)
  - Test automation (Quality mindset)
  - Quality first approach (Quality mindset)
  - Test specialisation (Quality mindset)
- Software Engineering
  - Unit testing (Quality mindset, skill)
  - Unit test coverage (Quality mindset, skill)
  - Code review (Quality mindset, psychological safety, collaboration)
  - Software design (Quality mindset, skill)
  - Pair programming (Psychological safety, collaboration)
  - CI/CD (Quality mindset, skill)
  - Tech talks (Skill, psychological safety, collaboration)
  - DevOps (Quality mindset, skill, agile mindset)
  - Cloud services (Skill)

6.2.3 Model: Employee and Team Characteristics

Each employee has the following characteristics:

- Skill - 0-1
- Experience - 1-10 for Software Engineers and QA Engineers, 5-15 for Scrum Masters and Product Owners
- Gender – 0 or 1
- Salary – a value in £
- Quality mindset – 0-1
- Collaboration - 0-1
- Flow - 0-1
- Estimation - 0-1
- Psychological safety - 0-1
- Agile mindset 0-1

The Team's characteristics are averages of the team members' characteristics.

6.2.4 Model: Randomness

A game without randomness is simply a simulation, and would not be interesting to replay.

All elements of the starting game state are randomised within certain bounds.

Employee characteristics:

- Experience
  - 1-10 years for Software Engineers and QA Engineers, 5-15 for Scrum Masters and Product Owners
- Gender
  - 70% male, 30% female (a somewhat realistic spread, but with enough chance of an employee being female to have some representation in the game)
- Salary
  - £25,000 - £50,000 for Software Engineers and QA Engineers, £10,000 more for Scrum Masters and Product Owners.
- Skill, Quality mindset, Collaboration, Flow, Estimation & Agile mindset
  - 0.2 - 0.8, with a median value of 0.5.
- Agile mindset for Scrum Masters and Product Owners
  - 0.6 - 0.8 to represent the additional Agile training they are likely to have had.
- Psychological safety
  - 0.1 - 0.5 to represent the lack of psychological safety when new teams are formed.

Project characteristics:

- Budget
  - £50,000 - £70,000
- Product Backlog
  - All User Stories are assigned a (hidden) effort value selected randomly from 1, 2, 3, 5 and 8.
- Customer
  - Quality mindset – 0.5 - 0.7

6.2.5 Simulation: Discovery of Software Development Practices

Discovery of Software Development Practices is the primary hidden goal of the game. Firstly, to teach the player about the practice, but more importantly to allow the player to experience the impact the practice has on the software development process.

Several discovery points were intended, but due to time constraints this was limited to Sprint Retrospectives. Other means of discovery are discussed in Future Work.

During a discovery event, the team learn about a new software development practice. Once discovered, this practice becomes a property of the project, becomes visible to the player and is available as a further learning opportunity (in the form of Retrospectives or Workshops).

6.2.6 Simulation: Retrospectives

This is an area where the model has been greatly simplified to aid player progress in the game. During a retrospective, the team discuss various topics and decide to make improvements in their use of 3 software development practices next sprint. The 3 practices are selected randomly, weighted towards practices they currently know the least about. This is to represent perhaps one individual on the team knowing about a specific practice and bringing it up for discussion during a retrospective.

Some further simplifications to the model to create a more coherent abstraction:

- In the early stages of the project the team can only discuss a subset of basic **Software Engineering** practices.
  - Tech talks, software design, unit testing, pair programming, CI/CD and code review.
- The team cannot discuss any **Agile** practices unless there is a **Scrum Master** on the team.
- The team cannot discuss any **Quality Assurance** practices unless there is a **QA Engineer** on the team.

Although the practices discussed in retrospectives are “discovered,” there is no guarantee that the team members will learn anything about these practices or improve their underlying characteristics relating to the practice. The learning process is discussed below.

6.2.7 Simulation: Learning

The game contains several opportunities for the team members to learn and improve their skills/characteristics:

Learning Opportunity	Outcome
Retrospective improvements	The software development practices discussed in retrospectives are mapped to 1 or more employee/team characteristic. For example: Unit testing --> *quality mindset, skill* Each team member has a chance to increase their characteristics in those areas, but only if their **agile mindset** is high enough. This represents the individual team member's *buy in* to the concept and practice of continuous improvement driven by the Agile retrospective. The calculation to determine if the team member's characteristic increases is simply: `Math.random() <= agileMindset` Where `agileMindset` is a value between 0 and 1.
Workshops	As CTO, the player may choose to run a workshop on any "discovered" software development practice. This causes a whole-day distraction for the team one day next sprint, but also increases all the team members' characteristics by a small amount.
Agile coaching	The player may choose to hire an Agile coach to spend time with the team during some of next sprint. This causes a small (randomly determined) distraction each day the coach is present, but may increase the team members' agile mindset. The chance of increase is randomly determined, and is compared to the Agile coach's skill or team member's agile mindset (whichever is highest). This is to represent the team member being able to extract useful value from the coaching. The amount of increase to the team member's agile mindset is dependent on the Agile coach's skill.

6.2.8 Simulation: Distractions & Firefighting

Several events cause distractions for the team. Their available effort (to work on items in the Sprint Backlog) is reduced.

- Retrospective improvements
    - The simulation here is somewhat simplistic. All team members are distracted a small amount (0.03) each day, and their available effort that day is reduced.
- Workshops
    - On the day the workshop is held, every team member's available effort is reduced to zero.
- Agile Coaching
    - On the days the Agile coach is present, every team member's available effort is reduced by a random amount determined when the Agile Coach is created (0.1 - 0.4).
- Firefighting
    - Firefighting represents the impact that a lack of concern for quality has on the project. As the project progresses, the likelihood of firefighting events increases *if the team have not improved their knowledge in some of the software development practices*. Every day, one of the software development practices that are related to the team's quality mindset characteristics is selected at random and its current value is compared against a random risk. The risk is reduced in the initial stages of the project to represent the greater complexity of systems and practices as the project progresses. If a firefighting event is generated, every team member is distracted that day by an amount between 0 and 1 of their available effort (calculated randomly for each team member). Improving the team's quality mindset reduces the risk of firefighting events during the project.

6.3 Simulation: The Sprint

The Sprint is the most complex area of simulation in the game. It is a sequence of the following states:

- CheckProjectStatusState
- RefinementState
- NavigationState -> ProductBacklogScene
- NavigationState -> SprintBoardScene
- NavigationState -> SprintReviewScene
- RetrospectiveState -> RetrospectiveScene

6.3.1 Check Project Status State

Here the end-game conditions are checked. If the 12 sprint are complete, or all the User Stories and bugs are done, the game ends and the game end scene is displayed.

6.3.2 Refinement State - User Story creation and estimation

Just prior to the new Sprint, more User Stories are defined, and the team estimate un-estimated stories at the top of the Product Backlog. The accuracy of these estimates is determined by the team's "estimation" attribute, and may vary slightly from the pre-determined effort required to complete a User Story. In this game, bugs are never estimated, in line with one school of thought in Agile that the

complexity of a bug is represented by the complexity of the original User Story. The story creation and estimation steps are hidden from the player, and simply occur in the background.

6.3.3 Refinement State - Velocity

Next the team's velocity is determined, derived as an average of the story points they delivered successfully in the previous 3 sprints. At the start of the game, before 3 previous sprints have occurred, random numbers between 25 and 35 are used instead. This number is then increased by an amount inversely proportional to the team's "estimation" attribute, to represent their level of inaccuracy. In early versions of the game this modification based on their ability to estimate could be positive or negative, but as the calculation of velocity has a tendency to decrease over the course of the game (explained further in Analysis) the estimation shift amount was fixed to a positive number.

6.3.4 Product Backlog Scene - Sprint Planning and creation of the Sprint Backlog

Once the velocity is determined, the Product Backlog is displayed to the user, and the features and bugs within the velocity limit are indicated with a red line. These items will become the Sprint Backlog, or the work the team will commit to during the Sprint. The player may reorder the backlog to indicate the priority of each user story or bug (as is normal in Agile practice). If there is a Product Owner on the team, they will move the customer's priorities to the top of the backlog automatically. If not, the player should do this (assuming they met with the customer to find out their current priorities).

The process of selecting the Sprint Backlog from the Product Backlog is automated because it is something that the Scrum Team do themselves (theoretically) without interference from stakeholders outside the team.

6.3.5 Sprint Board Scene - The Sprint Board

Next, the Sprint Board is displayed, and the player watches the progress of the team during the sprint. Behind the scenes a lot of processes are happening, most of which are not visible to the player.

At the beginning of each day during the sprint, each team member's available effort is calculated. It may be reduced depending on the following distractions:

- Retrospective improvements
- Workshops
- Agile Coaching
- Firefighting

Then, in turn each team member selects a User Story or bug to work on. Which item they choose depends on the team's "flow" attribute and the team member's "collaboration" attribute. Flow represents the behaviour of working on items in priority order to get the highest priority work completed first. If the team's flow attribute is high, they will work to complete the highest priority items. If it is low, they will start work on any item, and may have many items in progress at once. Collaboration represents the behaviour of working together on the same User Stories and bugs.

So, the team member selects an item based on the “flow” attribute and (if it is already being worked on by another team member) tests against their “collaboration” attribute to see if they are willing to work together. If the collaboration test fails, a different item is selected using the same process.

Once a team member has selected an item to work on, it will be moved to the Active column on the Sprint Board, and they will expend up to their available effort on the item.

In the process of working, the team member may write buggy code, depending on their “quality mindset” attribute. If they wrote buggy code, they will either add more effort to the bug they were working on, or a hidden bug (with a randomly determined effort) will be added to the project, if they were working on a User Story.

The process of selecting items, expending effort and producing bugs continues until all team members have expended all of their available effort for the day.

The team then have an opportunity to find any of the bugs that were created that day. Each bug is checked against the team’s “quality mindset” attribute and if successful the bug becomes visible and appears on the Sprint Board (and later the Product Backlog).

Where there are firefighting or workshop distractions happening that day, a dialogue is displayed to the user.

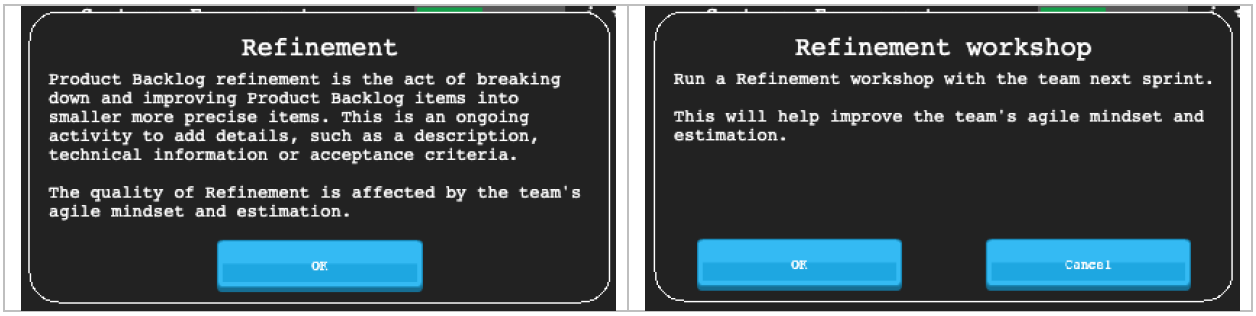
The day ends and a new one begins until the 10 days of the Sprint are complete.

At the end of the sprint, the customer has an opportunity to find hidden bugs (in the same way as the team), and the customer satisfaction is calculated based on whether their priorities were completed without bugs, and also based on the ratio of bugs to stories in the Product Backlog.

6.3.6 Player Learning

The core function of the game is to teach the player about good software development practices, how they can be used and how the work together to result in successful software development teams and projects. The mechanisms for teaching these practices are explicit instruction, exploration and insight, and repetition. Choosing between these three approaches to learning was one of the more difficult aspects of designing and developing the game. The game should be fun to play, so explicit instruction needs to be somewhat limited. The player also needs to understand the impact their choices and actions have on the game within the limited timeframe of 10 Sprints, so the ability to gain insights from exploration needs to be present without being so obvious that the player is spoon-fed information. Similarly, learning through repetition needs to be possible within the 10 Sprints of the game.

Explicit instruction is used in the Software Development Practices page where a player can choose to arrange Workshops for their team on one or more practices. The player is able to read about a specific practice and the characteristics improved by learning this practice are also indicated to the player.



Exploration and insight are used throughout the game as the primary means of understanding the impact of different software development practices, and the overall impact of the three categories of practices (Agile, Quality Assurance, Software Engineering). For example, the majority of test subjects understood that making improvements to the team's knowledge on Quality Assurance practices led to fewer bugs.

Repetition was used as the main way to teach the player about standard Agile practices (specifically Scrum). Scrum events repeated with a regular cadence as they do in real life software projects. Repetition combines with exploration and insight to help players understand the impact of other choices on repeating sprints. The player is able to see the impact on tea efficiency, bug creation or customer satisfaction at repeated intervals throughout the game.

7 Evaluation

7.1 User Assessment Methodology

To measure the success of the project, test subjects played the game and their learning responses were estimated via their success within the game and a post-game survey. The game results were collected via a screenshot of the Game Over screen.

The design of the survey focussed on whether the test subjects felt the game was a useful learning tool, whether they learned more about the 23 software development practices described earlier and how these practices impact the success of a software development project. The survey included questions asking the subject to give weighted answers, multiple choice answers and extended text answers.

The test subjects were asked to watch a short video, explaining how the game worked. The subjects were asked to play the game and fill out the survey once complete.

The instruction video can be found here: http://project-genesis-a-software-development-fantasy.s3-website-eu-west-1.amazonaws.com/welcome.html

7.2 Survey Questions

1. Were you able to play the game from beginning to end (the Game Over screen)?
2. Please upload a screenshot of the Game Over screen. (file upload)
3. On a scale of 1-5, how much do you feel following 3 areas of software development were addressed in the game?

   a. Agile
   b. Quality Assurance
   c. Software Engineering
4. Which of the following 23 software development processes/practices do you feel were addressed in the game?
   a. Three Amigos
   b. Refinement
   c. Daily Scrum
   d. Sprint Retrospective
   e. Sprint Review
   f. Psychological Safety
   g. WIP Limit
   h. Sprint Goal
   i. Continuous Improvement
   j. Customer Engagement
   k. Test Design
   l. Test Automation
   m. Quality First Approach
   n. Test Strategy
   o. Unit Testing
   p. Test Coverage
   q. Code Review
   r. Software Design
   s. Pair Programming
   t. CI/CD
   u. Tech Talks
   v. DevOps
   w. Cloud Services
5. What do you feel you learned from the game about how the three categories of processes/practices (Agile, Quality Assurance and Software Engineering) impact the success of a software project?
6. Which of the following roles did you hire during the game?
   a. Software Engineer
   b. QA Engineer
   c. Scrum Master
   d. Product Owner
   e. Agile Coach
7. Pick one of the roles above, and describe their impact on the success of the project in your game.

8. Do you agree or disagree that running the following workshops had the effect claimed (in later sprints)?
    a. Learning about Software Design improved the quality and amount of work done.
    b. Learning about Unit Testing slowed the team down.
    c. Learning about Code Review helped the team collaborate better.
    d. Learning about Refinement improved the team's ability to estimate User Stories.
    e. Learning about WIP Limits led to more stable, predictable flow of work in Sprints.
    f. Learning about Test Design led to fewer bugs being created.
    g. Learning about Test Automation led to fewer undiscovered bugs.
    h. Learning about Tech Talks improved the psychological safety of the team.
9. What do you think was the main effect of improving the team's software engineering practices, such as Software Design, CI/CD, Cloud Service, DevOps and Unit Testing?
10. What do you think was the main effect of improving the team's quality assurance practices, such as Test Design, Test Automation, Quality First Approach and Test Strategy?
11. What do you think was the main effect of improving the team's Agile practices, such as Refinement, Daily Scrum, Sprint Review, Sprint Retrospective, Sprint Goal, Psychological Safety, Three Amigos, WIP Limit, Continuous Improvement and Customer Engagement?
12. Did you realise that running workshops, hiring an Agile coach and the outcomes of retrospectives all helped improve the team members' attributes? The attributes were: Skill, Quality Mindset, Flow, Estimation, Psychological Safety and Agile Mindset.
13. What was the best way to avoid firefighting events?
    a. Fix all the bugs in the backlog.
    b. Improve the team members' Agile Mindset.
    c. Learn about Quality Assurance practices.
    d. Hire more Software Engineers.
14. What advice would you give to a new player to help them "win" the game? In other words, what do you think is the winning strategy?
15. Is the game representative of software development in the real world?
16. As a tool to learn about running successful software projects and teams, how do you think the game could be improved?

7.3 Limitations

There are some important limitations which may have affected the test subjects' answers and/or the validity of the final results.

- Realism
    - The game attempts to strike a balance between a realistic simulation to teach real-world software development and a contrived, exaggerated, and simplified representation within a game (both for playability and to include all the necessary elements within a limited timeframe). The whole game is played over a 20 week period (10 2-week Sprints), so the speed that employees learn and grow was accelerated. The range of

opportunities for employees to learn and grow was condensed to just 3 (Retrospectives, Workshops and Agile coaching).

- Randomness
    - There is necessarily a lot of randomness in the game. This has been constrained in many areas (such as generating attribute values between certain bounds) after lengthy playtesting, but it is possible that a test subject could have started an "unwinnable" game due to the random starting conditions and inexhaustive playtesting.
- Test Subject Bias
    - All of the test subjects are known to me in a work context, and I have managed or overseen 10 out of the 11 test subjects. While I believe the subjects' answers are truthful, there may be some positive bias in the answers due to both my personal and working relationships with the subjects.
- Test Subject Knowledge
    - The test subjects are all either Software Engineers or QA Engineers, ranging from junior to lead in position. The subjects in senior and lead positions probably already have more software development knowledge than the game's intended audience.
    - About half of the test subjects have been part of an Agile transformation process, so will have been exposed to more information about successful Agile practices that the game's intended audience.
    - All of the test subjects work in software development, and there is a risk that they answered some of the survey questions based on their pre-existing knowledge about software development rather than their experience in the game.
- Learning Opportunities
    - The learning opportunities presented by the game are partly dependent on how well the player is doing. Certain learning opportunities may not have been presented to the test subject if they didn't perform certain actions. For example, Quality Assurance practices won't be raised in Retrospectives without a QA Engineer on the team.

7.4 Survey Results

7.4.1 Questions 1 & 2 - Game completion

All 11 test subjects were able to complete the game.

7.4.2 Question 3 - Categories of software development practices addressed in the game

On a scale of 1-5, how much do you feel following 3 areas of software development were addressed in the game?

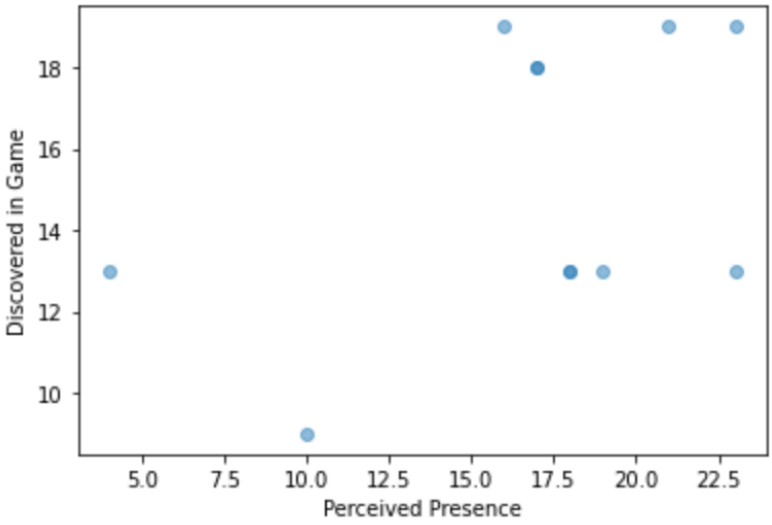
	Min	Max	Average
Agile	3	5	4.18
Quality Assurance	3	5	4.09
Software Engineering	2	5	4.00

With 1 exception, all answers given showed a fairly similar number of 3s, 4s and 5s. Only 1 test subject answered 2. Broadly, this shows that the game addressed these 3 areas sufficiently well for the players.

7.4.3 Question 4 - Software development practices addressed in the game

Which of the following 23 software development processes/practices do you feel were addressed in the game?

Here there was a wide variation in the number of practices selected, from 4 to 23, though the majority of answers clustered around the average of 16.9, with the 25th percentile being 16.5 and the 75th percentile 20.0. There was a weak but visible correlation between the number of practices that the player felt were addressed in the game and the number of practices their team discovered during the game.



5 of the practices were selected by 10 of the test subjects, and a further 1 practice was selected by all 11. These were:

- Sprint Retrospective - 10
- Sprint Review - 10
- Unit Testing - 10
- Pair Programming - 11
- CI/CD - 10
- Cloud Services - 10

6 of the practices were available to discussion in Retrospective from the beginning of the game. As the game progresses, more practices become available. There is little correlation between these early practices and the practices the players felt were addressed by the game. 3 were selected by 10 or more test subjects. The early practices were:

- Tech Talks - 9
- Software Design - 8
- Unit Testing - 10
- Pair Programming - 11
- CI/CD - 10
- Code Review - 8

The remaining 3 early practices were selected by 8 or more of the test subjects. Because the practices addressed in each player's game were not recorded, it is difficult to know if the practices were simply not presented to the player due to the random nature of the game (although unlikely), or whether the player did not realise or remember that the practices were discussed by their team in retrospectives or learned about in Workshops.

7.4.4 Question 5 - Perceived learning

What do you feel you learned from the game about how the three categories of processes/practices (Agile, Quality Assurance and Software Engineering) impact the success of a software project?

The main theme in the answers given to this question was the importance of improving all 3 areas for success. This was stated by 5 of the 11 test subjects explicitly, and a further 3 subjects discussed the importance of improving all 3 areas separately.

One of the most important lessons of the game was that all 3 categories of software development practices are equally important to the success of software projects and the software teams working on them. I think the answers to this question make it clear that the majority of test subjects walked away from the game understanding this well.

Some additional insights from the answers:

- That it can be hard to balance the need of customers and bugfixes/general improvements.
- Early sprints were chaotic, with many bugs created and few features completed. Improving the quality of practices improved this in later sprints.

7.4.5 Question 6 - Roles hired

Which of the following roles did you hire during the game?

Role	*Number of players who hired this role*
Software Engineer	*10*
QA Engineer	*10*
Scrum Master	*6*
Product Owner	*7*
Agile Coach	*8*

All players hired at least 1 role.

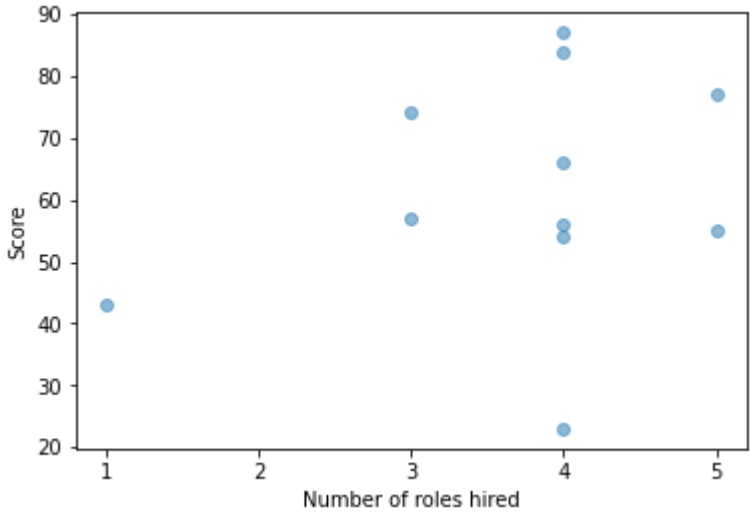
Robert Hill

Student number: 189047532

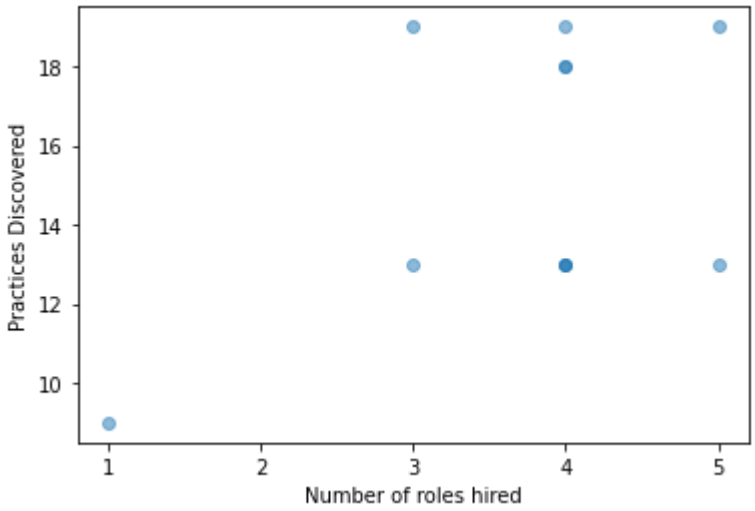
Student ID: rdh36

There was a general positive correlation between the number of roles that were hired (by a single player) in the game and both the final score and the number of software development practices discovered during the game.

Number of roles hired against final score:

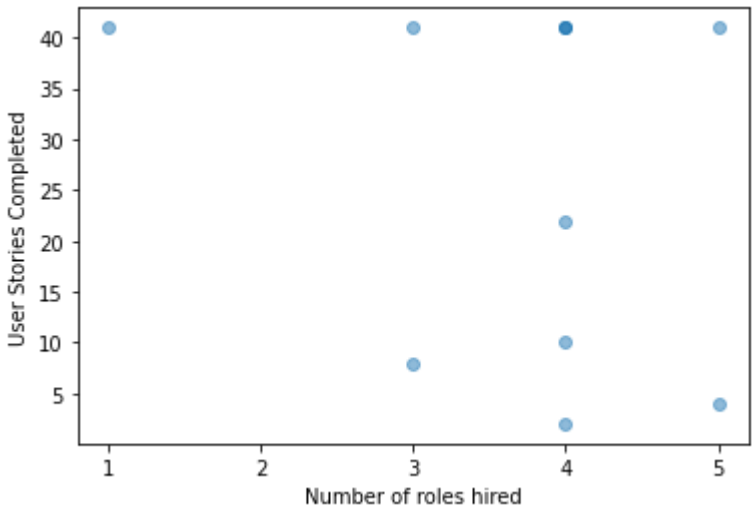


Number of roles hired against number of practices discovered:



There is a possible correlation between the number of roles hired and the number of User Stories completed by the end of the game, though this is less clear from the available data.

Number of roles hired against number of User Stories completed:



Taken together, these three graphs show that players generally performed better when they hired a broader range of the specialised roles available in the game.

7.4.6 Question 7 - Role analysis

Pick one of the roles above, and describe their impact on the success of the project in your game.

8 of the test subjects responded that hiring a QA engineer caused a reduction in bugs for the remainder of the project. It is notable that the majority of players completed the project successfully. No players ended the project with any "hidden" bugs (bugs in the software but not represented in the backlog) and only 2 of the players had incomplete bugs in the backlog at the end of the game. This is one area where tweaks to the game model configurations to improve realism may have had the unintended consequence of making QA Engineers too good at finding hidden bugs, and decreasing the team's likelihood of creating new bugs, making the game too easy in this area.

2 test subjects correctly identified that the Product Owner handled the prioritisation of customer needs.

None of the test subjects discussed the effects of hiring a Scrum Master or an Agile Coach.

All of the statements made by the test subjects about the effect of hiring QA Engineers and Product Owners were correct, meaning that the players correctly inferred from the game what impact these roles had.

The responses to this question also highlighted 2 other issues or outliers.

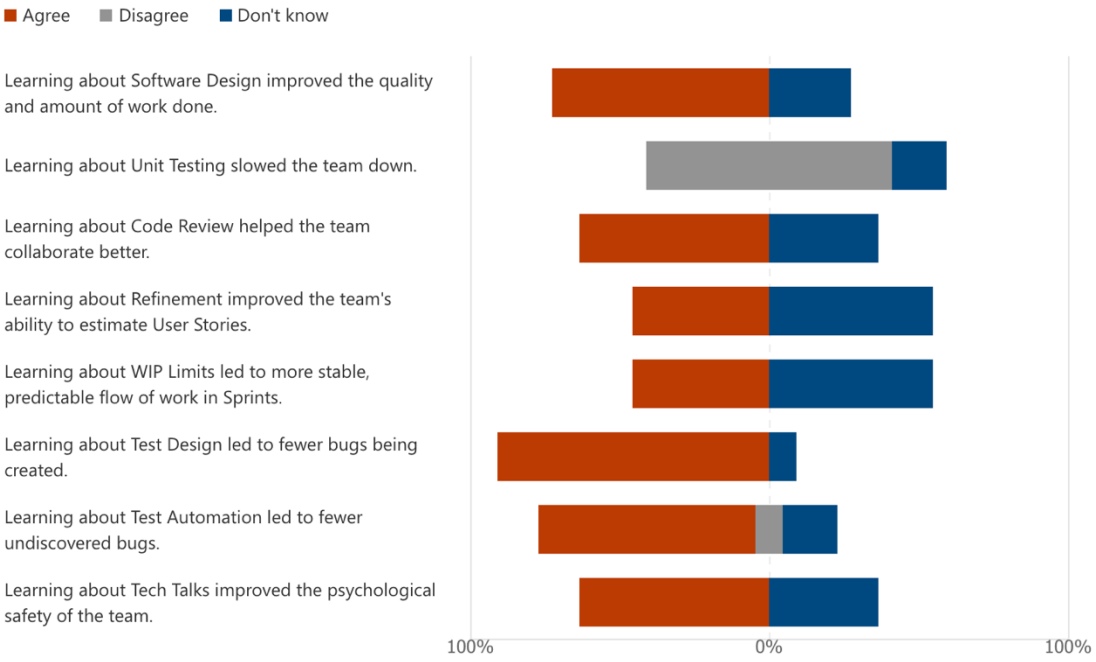
One player was unable to overcome the number of bugs being created, even though they had hired a QA Engineer. They were only able to complete 2 stories during the whole project and appear to have spent the time mired in bugs.

One player hired 2 additional Software Engineers and no other roles. They completed the project early, but ended with a low score and few practices discovered.

These outliers suggest further work is needed to ensure the boundaries of success and failure are better understood and if necessary restricted.

7.4.7 Question 8 - Effects of Workshops

Do you agree or disagree that running the following workshops had the effect claimed (in later sprints)?



Excluding “Don’t know” responses, in all cases more test subjects agreed/disagreed in accordance with how the game functioned, with only 1 answer given that broke from this. Some answers show a large number of “Don’t know” responses, suggesting that the players weren’t able to infer correctly from the game in certain instances. There is a risk that players weren’t responding based on their experience in the game here, but instead based on their understanding of software development.

7.4.8 Question 9 - Effect of improving Software Engineering practices

What do you think was the main effect of improving the team's software engineering practices, such as Software Design, CI/CD, Cloud Service, DevOps and Unit Testing?

The primary effect of improving these practices is to increase the team members’ “skill”. This increases the amount of work they can complete each day. However, 4 of the practices given as examples in the question also increase the team members' “quality mindset”. This is reflected in the test subjects’ answers to this question. Most of the test subjects felt that improving software engineering practices led to improvements in quality, speed of delivery, or both. 3 of the test subjects mention only improvements to quality, suggesting that they were unaware that improving these practices also increased the efficiency of the team.

7.4.9 Question 10 - Effect of improving Quality Assurance practices

What do you think was the main effect of improving the team's quality assurance practices, such as Test Design, Test Automation, Quality First Approach and Test Strategy?

Almost all of the test subjects correctly identified that fewer bugs were created as a consequence of improving these practices (in other words, the team wrote less buggy code). Two of the test subjects also correctly identified that bugs were identified and raised more quickly by the team, reducing the number of bugs left to be found by the customer. The majority of test subjects commented that the reduction in bugs enabled the team to focus on delivering customer features in later sprints.

None of the test subjects identified here that improving the team's quality assurance practices led to a reduction in firefighting events.

7.4.10 Question 11 - Effect of improving Agile practices

What do you think was the main effect of improving the team's Agile practices, such as Refinement, Daily Scrum, Sprint Review, Sprint Retrospective, Sprint Goal, Psychological Safety, Three Amigos, WIP Limit, Continuous Improvement and Customer Engagement?

The test subjects gave a wide range of answers here, often suggesting that they didn't understand the impact of improving these practices.

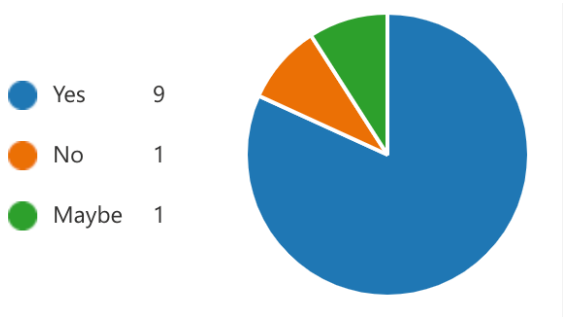
The main effect of improving Agile practices was to increase the team members' "agile mindset" attribute, which increases the chance of the team learning from Retrospective actions and Agile coaching. Only 2 of the test subjects responded in a way that suggested they understood this effect.

However, 5 of the 10 practices are also connected to the team members' "collaboration" attribute. Increasing this attribute makes the team more likely to work together, and 4 of the test subjects did notice and comment on this effect.

7.4.11 Question 12 - Effect of learning opportunities

Did you realise that running workshops, hiring an Agile coach and the outcomes of retrospectives all helped improve the team members' attributes?

The attributes were: Skill, Quality Mindset, Flow, Estimation, Psychological Safety and Agile Mindset.



Player understanding of Team learning opportunities.

The answers suggest that the test subjects understood these 3 discovery events led to improvements in team members' attributes. However, the wording of the question here may have led test subjects to answer affirmatively.

7.4.12 Question 13 - Avoiding Firefighting events

What was the best way to avoid firefighting events?

The responses were split between 2 of the available answers.

- Improve the team members' Agile Mindset. (3)
- Learn about Quality Assurance practices. (8)

The correct answer here was to improve QA practices.

7.4.13 Question 14 - Winning strategy

What advice would you give to a new player to help them "win" the game?

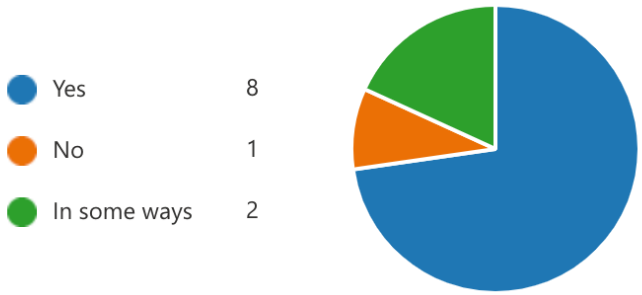
In other words, what do you think is the winning strategy?

There was a range of answers to this question, but 4 general themes emerged from the test subjects' answers:

Strategy	Number of test subjects
Improve software development practices early in the game.	5
Hire all available roles on the team.	4
Hire a QA Engineer early in the game.	2
Hire an Agile Coach early in the game.	2

7.4.14 Question 15 - Realism

Is the game representative of software development in the real world?



7.4.15 Question 16 - Improvements

As a tool to learn about running successful software projects and teams, how do you think the game could be improved?

There are no common themes to the answers given here, but they provide some useful feedback on improvements that could be made in future versions.

- I should have hired straight up. I assumed the budget I was allocated at the start was for the whole project so I was reluctant to burn it up front
- Depends on the audience, but perhaps somebody might not be familiar with roles such as Product Owner and what the role is responsible for. Having a brief information about such roles could potentially lead to different choices when building the team.
- Have the customer change requirements 50% of the features halfway through the game.
- Less obvious pointers on what would improve the success of the team from the retros so more thought is required
- Rather than having the Manager say "we didn't complete this goal, what happened?" or "it's great that we did this", show the team's discussions regarding what helped them achieve the goal or what delayed them. This would make it clearer which aspects of the game are most helpful and should be continued each sprint, and which aspects are hindering the team's progress and how to best avoid those.
- Probably the ability to "block" due to other tasks or external dependencies. Team members holidays and/or sick days.
- We should be able to view the whole Product Backlog at any given point in time.
- Unless I missed it, an explanation of the WIP limit would have been useful as I was initially unsure of what it was. Perhaps a little insight into the agile ceremonies in real-time would have helped to understand the dynamics of the team and their interactions, leading to better decisions when playing the next turn of the game. While I thought I knew about Agile practices to a certain degree from my previous experience of them, each company/team has their own idea and implements them differently so being able to see the interactions and how they develop over time would be most useful to me.
- I think of post-analytics chart, and tips on where things could have been improved would be a valuable asset.
- A way to influence the customer viewpoint and view previous customer feedback A timeline based sprint control. The sprints, once kicked off, were set in stone and there was no way to influence the outcome beyond that of a voyeur....whereas in real-world the outcome can be shaped?
- Clarity on effect of team introducing/implementing new processes and how that directly impacts team effectiveness/success. Perhaps percentage knowledge bars shown with the sprint board, player is able to review the sprint board at the end of the sprint in their own time, details of burndown etc.

Robert Hill

Student number: 189047532

Student ID: rdh36

8 Analysis

8.1 Implementation

The core game model simulates the software development process accurately enough to appear realistic to a player. Someone working in the industry would recognise the flow and cadence of the Sprint, the movement of user stories and bugs across the sprint board, the impact distractions have on the ability to do work, the changing priorities of customers, the availability (or lack of) suitable candidates to hire or the budgetary restrictions that must be managed.

This semi-realistic simulation is suitable for the purpose of teaching software development practices, but it differs from the reality of software development in several key areas.

1. The impact of learning is exaggerated and accelerated to have an impact on gameplay. Without this exaggeration, the improvements in the skills of team members would not result in work being done more efficiently and to a higher standard of quality by the end of the game.
2. The Sprint Retrospective is simplistic and does not reflect how teams that don't understand or value retrospectives would use it. Teams with a poor grasp or appreciation of Scrum and Agile would not improve via the retrospective in the way they do in the game.
3. Company run workshops are less common in the real world. In practice, most Software Engineers learn from a variety of sources, and very often learn new things as and when required by the job. It was important to provide a means of directed learning so that the player could control the areas in which the team learn new practices.
4. Software development practices are improved indirectly by learning similar software development practices. While this may be broadly true in some areas, it may not be realistic in all cases. For example, if the team spend time in a Test Design workshop, their "quality mindset" characteristics will increase. This will increase the team's knowledge of Test Design and Test Automation simultaneously.
5. Team members' learning opportunities are limited to retrospective actions, workshops and coaching. In reality, many more learning opportunities are available. This is due to the time constraints of the project rather than a simplification for playability.
6. The calculation of the team's velocity and commitment each Sprint seems unrealistic and has a tendency to decrease over time if the Scrum method of averaging recent velocities is used, even when boosted by a percentage each sprint. It doesn't reflect how teams choose their own commitment, or how their choice is influenced by other factors. In extreme cases this can result in the team taking on significantly less work than they are capable of completing in a Sprint.
7. The player is unable to affect the team's commitment each Sprint. While this is in line with Scrum best-practice, it is unrealistic in the real world and also limits player agency in the game.
8. The impact of diversity on a Scrum team is missing from the game. This is due to the time constraints of the project rather than a simplification for playability.
9. The happiness of team members is missing from the game. This is due to the time constraints of the project rather than a simplification for playability.

10. Resignations or redundancies are missing from the game. This is due to the time constraints of the project rather than a simplification for playability.
11. There is no penalty for repeatedly ignoring the customer's priorities. While the customer makes their displeasure known to the team during the Sprint Review, there is no impact on the game.

The game suffers from some problems with playability and player agency.

Many of the processes are hidden from the player, as they would be in the real world. It is, for example, not possible for the player to observe the inaccuracy of estimates, the exact relationship between the team's approach to quality and the resulting firefighting events, how the remaining effort on user stories is reduced and how and why bugs are created. This adds some realism, but hampers the player's ability to understand how their choices and actions positively or negatively affect the outcome of the game.

Similarly, the initial random state of the game (included to add realism) can sometimes cause the game to be too difficult to play. In particular, if all team members begin the game with low "skill", "quality mindset" and "collaboration" attributes, the outcome is likely to be that they are unable to complete mush work on User Stories and create large numbers of bugs each sprint.

There is no way for the player's existing knowledge of software development to be represented. Importing player knowledge into the game state could offer good opportunities for differing starting conditions or branches in gameplay, but would have added a lot of complexity to the testing and playtesting of game outcomes.

The game's reward system consists of:

- Visual feedback in the form of the sprint board and burndown chart.
- Verbal and visual feedback from the customer each Sprint Review.
- Interruptional feedback in the form of firefighting events.
- Visual feedback in the form of a notification when player characteristics improve.
- Project end feedback in the form of project delivery success.
- Project end feedback in the form of software development practices learned.

These rewards appear to be enough to maintain player engagement and if a player's actions are having a positive or negative effect.

8.2 Game Results

This section concerns the success or failure of the player as defined by their results on the Game Over screen at the end of the game.

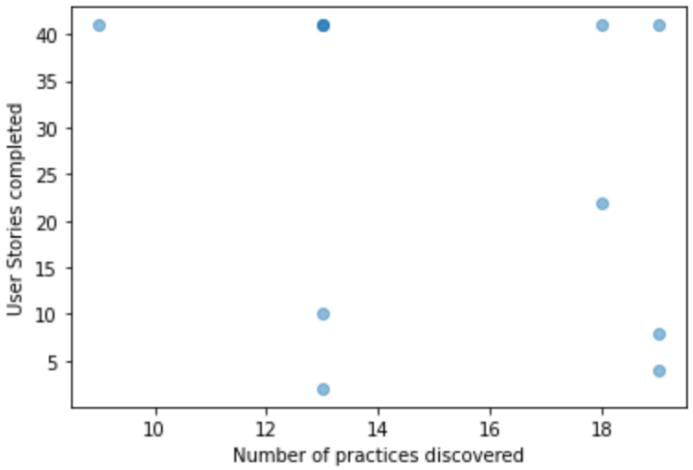
The Game Over screen.

During development some parameters of the game model were repeatedly tuned to improve the apparent realism of the simulation. To counteract too many bugs being created, some last-minute changes were made to reduce the rate of new bug creation. The majority of the test subjects were not representative of the game's target audience, with many of them having prior knowledge of the importance of various software development practices. For these two reasons, many of the subjects performed better than expected. However, there was still some wide variation in outcomes.

- 5 subjects completed the project early.
- 1 subject completed the project on time.
- 5 subjects did not complete the project on time.

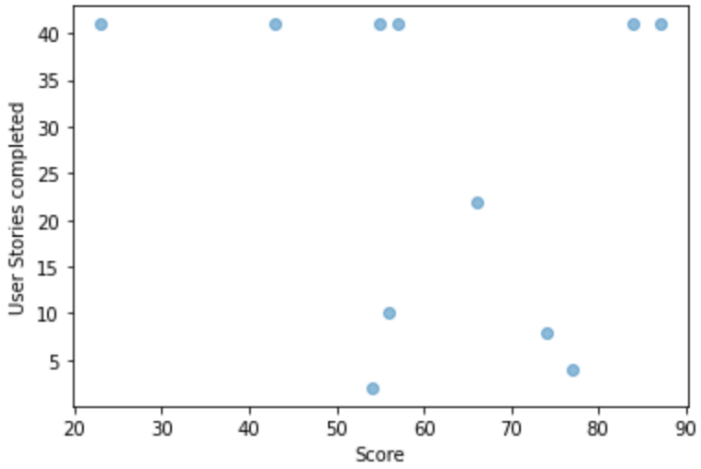
Of the subjects who failed to complete the 41 User Stories in the backlog before the end of the 10 Sprints, the number of stories completed ranged from 2 to 22, with a mean average of 9.2.

The number of software development practices discovered by each player ranged from 9 to 19, with a mean average of 15.18. There was no obvious relationship between the number of practices discovered and the success of the project (defined by completing the stories on time).



Stories completed against discoveries.

The overall score (a measure of how much knowledge the team had in all 23 software development practices) ranged from 23 to 87. Similarly, there is no obvious relationship between the overall score and the success of the project.



Stories completed against overall score.

The player results highlight that further work is required to refine the parameters of the model and simulation to more closely meet the needs of the intended audience.

8.3 Player Survey

The player survey shows that the game was clearly successful in two key areas.

1. The majority of players felt that the game is representative of software development in the real world.
2. The game was an effective learning tool for the majority of players.

3. The majority of players understood that success required improvement in all 3 areas of the software development process.

On the first point, while there is scope for improving the parameters of the model and simulation, the mechanisms by which the simulation functions appear realistic to the player.

The second point is interesting because the test subjects clearly felt that the game was useful as a learning exercise even though many of them already had good software development knowledge.

Point three is significant because this is a fundamentally important thing to learn about successful software projects. It may not usually be considered by most team members, who are not viewing the project from the outside looking in and not often able to see the project at the right scale to appreciate this point.

However, the survey also shows differing responses from individual respondents. One respondent felt that the causal relationship between Workshops and the team learning they led to was too obvious to the player. Another respondent seemed not to spot most of the learning opportunities in the game (possibly because they spent the game mired in bugs, unlike the rest of the test subjects).

8.4 The Game as a Learning Tool

As the project survey indicates, players of the game walked away with a better understanding of the importance of Software Engineering, Quality Assurance and Agile practices, and how they need to be balanced for success. Players learned and understood that appropriate improvements to teams and the practices they use are an investment in the software development process, leading to greater quality and stability of delivery. Due to the interactive and fun nature of the game, the players learned through experience in a way that they may not have via other sources.

Whilst the test subjects felt that the game was an effective learning tool, it is not possible to understand a great deal about what they learned based on the survey responses, or to understand for which of the test subjects it was most effective and why.

The test subjects seemed to understand the impacts of learning Quality Assurance and Software Engineering practices better than they understood the impacts of learning Agile practices. One subject responded that more information about Agile would be useful:

> “Perhaps a little insight into the agile ceremonies in real-time would have helped to understand the dynamics of the team and their interactions, leading to better decisions when playing the next turn of the game. While I thought I knew about Agile practices to a certain degree from my previous experience of them, each company/team has their own idea and implements them differently so being able to see the interactions and how they develop over time would be most useful to me.”

Several of the test subjects commented informally on how they felt the game was a great learning tool and about specific practices with which they were unaware, but these remarks weren’t recorded as part

of the survey, and any test subjects who had a negative view of the game as a learning tool are unlikely to have come forward with their views outside of the survey.

9 Future Work

There are several areas with more detail, complexity or other changes could be added to the game model to add realism and potentially improve learning opportunities.

9.1 Intended Audience

The most useful next step would be to gather feedback from players more alike the intended audience, with a more detailed player survey to understand exactly what the players are learning and why. Computer Science undergraduates and junior software engineers are likely to be the most suitable end user for this game, and gathering more feedback from that audience, plus further feedback from other groups with more or less knowledge about software development practices would help understand how the game model parameters could best be configured.

9.2 Player Learning

The current version of the game is the first version where feedback from play testers was possible. As such it was the first point at which the effectiveness of the 3 approaches to learning (explicit instruction, exploration and insight, and repetition) could be evaluated. There is a lot for scope for improvement in this areas. Overall, the test subjects found the game easier to play than expected, and some feedback suggested that too much information was given about the positive effects some actions would have, limiting some opportunities for exploration and insight. Versions of the game could be produced which remove some of the explicit information about the impacts of improving the different software development practices. Further play testing could be used to find the sweet spot for players with different levels of knowledge about software development, and the difficulty levels could be introduced which present different amounts of explicit instruction.

9.3 Diversity

Gender, age, ethnic origin and social class could be added as employee attributes to create a derived “diversity” attribute for the team, perhaps using a Euclidian distance calculation. This could be used in combination with “agile mindset” and “psychological safety” to better model the Sprint Retrospective. Teams that do better retrospectives learn and adapt more quickly than teams that don’t. Diverse teams are likely to generate more diverse ideas and discussions in their retrospectives.

9.4 Player Knowledge

The player’s existing knowledge could be imported into the game via mini-games, such as matching concepts to categories or descriptions. This would allow the player to focus on learning about areas of software development that they don’t already know.

9.5 Additional Discovery Points

In-game discovery points are opportunities for the game’s team members to learn software development topics. The game would benefit from a range of additional discovery points, both for realism and to add interest, variation and choice to the player. Some possible additions:

- Hiring other types of consultants (in addition to Agile Coaches)

- Employee training courses
- Conference attendance
- Local tech meet-ups

9.6 Randomness

Randomness could be limited in some areas to improve playability. In particular, the player could choose from an easy, medium or hard company at the start of the game, which would generate a team with random characteristics within stricter bounds.

9.7 Resignations, redundancies & happiness

While the player is able to hire candidates that are available, they aren't able to fire any of their team. This would be a simple way to give the player more agency.

Similarly, the team members are currently unable to leave, no matter how badly the project is being run. Adding a "happiness" attribute to each team member, affected by events in the game would make it easy to model team member resignations in response to poor conditions or pay.

9.8 Agile

The inclusion of Scrum in this game isn't intended to suggest that Scrum is the right way to do software development. The important point is having and improving on a development process that encourages continuous feedback loops and continuous process improvement. Enabling the player to explore other practices, such as Kanban, Lean or even Waterfall could be a useful addition to the game, and provide further learning opportunities.

9.9 Estimation, Velocity and Sprint Planning

The way the team's commitment each sprint is calculated, based on their previous velocity and adjusted by their ability to estimate well, does not produce realistic values. Further work could focus on finding more appropriate ways to model this or allowing the player to adjust it manually.

Allowing the player to adjust the team's commitment each Sprint, by putting pressure on the team to do more, would add realism and player agency to the game. This could also impact employee happiness and make them more likely to quit their job.

9.10 Agile Mindset

The use of the "agile mindset" attribute could be expanded as a restriction on learning from retrospective outcomes, and also the distraction they cause. In this way, a team member with a low "agile mindset" would show little interest in retrospectives and the actions that the team have decided to perform. They would be less distracted by the retrospective actions, but also would not learn much from them. This would be a difficult addition and could have a negative impact on playability, but would introduce the player to a common problem on some teams.

9.11 Customer Satisfaction

Customer satisfaction is indicated in 2 ways.

- Their satisfaction with their priorities being worked on and the overall quality of the product at the end of each sprint.
- Their satisfaction with the number of features complete and the overall quality of the product at the end of the project.

The two are no linked. Improvements could be made here to indicate the ongoing/cumulative satisfaction of the customer at the end of the game. Additionally, the game could end early if the player consistently fails to satisfy the customer, representing the customer leaving and taking their business elsewhere.

9.12 Player observation of “hidden” areas of the model

Providing mechanisms for the player to get a view of some hidden areas of the model could improve the sense of player agency and aid the player in understanding the impact of their actions. Some examples:

- Understand the quality of the codebase by buying a static analysis tool.
- Employ a consultant to perform a quality or Agile audit.

Robert Hill

Student number: 189047532

Student ID: rdh36

10 Conclusion

Project Genesis has been a great success in a number of areas.

The game is interesting and fun to play, offering a genuine challenge to the players while maintaining their interest to the end.

The simulation of the software development process is realistic enough to give the feel of a “real world” software project. It models the activities of a software team with varying skills and abilities well, from a team mired in bugs and constantly firefighting at one extreme, to smooth and stable development of features at the other. It is flexible enough in its current state to be modifiable with simple configuration changes to increase or decrease difficulty by changing capabilities of team members, changing relative probabilities or risks, changing budgets or project length. It is extensible to allow future development to add more realism and variety to the model and simulation.

Most importantly, the project succeeded in its goal of creating a learning tool to promote the understanding of software development best practices, what those practices are and how to use them together. As the project survey indicates, players of the game walked away with a better understanding of the importance of Software Engineering, Quality Assurance and Agile practices, and how they need to be balanced for success. Players learned and understood that appropriate improvements to teams and the practices they use are an investment in the software development process, leading to greater quality and stability of delivery. Due to the very visual feedback, and interactive and fun nature of the game, the players learned through experience in a way that they may not have via other sources.

With a small amount of additional work to better understand the target audience, tune the model parameters to match that audience, and ensure the game runs cross-browser, the game could be deployed without further modification as an educational tool and/or commercialised.

There is huge scope for ongoing development, adding more elements of the software development process to the simulation, adding more software development practices to learn about, adding more player learning mechanisms to the game and more depth to the topics learned by the player.

Robert Hill

Student number: 189047532

Student ID: rdh36

11 References

[i] Javdani, Taghi et al. “The impact of inadequate and dysfunctional training on Agile transformation process: A Grounded Theory study.” *Inf. Softw. Technol.* 57 (2015): 295-309.

[ii] J. M. Rojas, T. D. White, B. S. Clegg and G. Fraser, "Code Defenders: Crowdsourcing Effective Tests and Subtle Mutants with a Mutation Testing Game," *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, 2017, pp. 677-688, doi: 10.1109/ICSE.2017.68.

[iii] *Maren Krafft, Gordon Fraser, and Neil Walkinshaw. 2020. Motivating Adult Learners by Introducing Programming Concepts with Scratch. In Proceedings of the 4th European Conference on Software Engineering Education (ECSEE '20). Association for Computing Machinery, New York, NY, USA, 22–26. DOI:https://doi.org/10.1145/3396802.3396818*

[iv] Vos, Tanja & Prasetya, Wishnu & Fraser, Gordon & Martinez-Ortiz, Ivan & Perez-Colado, Ivan & Prada, Rui & Rocha, Jose & Silva, António. (2019). IMPRESS: Improving Engagement in Software Engineering Courses through Gamification.

[v] J. Hamari, J. Koivisto and H. Sarsa, "Does Gamification Work? -- A Literature Review of Empirical Studies on Gamification," *2014 47th Hawaii International Conference on System Sciences*, 2014, pp. 3025-3034, doi: 10.1109/HICSS.2014.377.

[vi] Gresse von Wangenheim, Christiane & Savi, Rafael & Borgatto, Adriano. (2013). SCRUMIA—An educational game for teaching SCRUM in computing courses. Journal of Systems and Software. 86. 2675-2687. 10.1016/j.jss.2013.05.030.

[vii] Torgeir Dingsøyr, Sridhar Nerur, VenuGopal Balijepally, Nils Brede Moe. 2012. A decade of agile methodologies: Towards explaining agile software development, Journal of Systems and Software, Volume 85, Issue 6, 1213-1221, https://doi.org/10.1016/j.jss.2012.02.033.

[viii] A. Rauf and M. AlGhafees, "Gap Analysis between State of Practice and State of Art Practices in Agile Software Development," 2015 Agile Conference, 2015, pp. 102-106, doi: 10.1109/Agile.2015.21.

[ix] Lehtinen, T.O.A., Itkonen, J. & Lassenius, C. Recurring opinions or productive improvements—what agile teams actually discuss in retrospectives. Empir Software Eng 22, 2409–2452 (2017). https://doi-org.ezproxy3.lib.le.ac.uk/10.1007/s10664-016-9464-2

[x] Matthies, Christoph & Dobrigkeit, Franziska & Ernst, Alexander. (2019). Counteracting Agile Retrospective Problems with Retrospective Activities. 10.1007/978-3-030-28005-5_41.

[xi] M. Paasivaara, "Teaching the Scrum Master Role using Professional Agile Coaches and Communities of Practice," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, Madrid, ES, 2021 pp. 30-39.doi: 10.1109/ICSE-SEET52601.2021.00012keywords: {training;tools;software;personnel;scrum (software development);interviews;software engineering}url: https://doi.ieeecomputersociety.org/10.1109/ICSE-SEET52601.2021.00012

[xii] *Maria Paasivaara and Casper Lassenius. 2016. Challenges and Success Factors for Large-scale Agile Transformations: A Research Proposal and a Pilot Study. In Proceedings of the Scientific Workshop Proceedings of XP2016 (XP '16 Workshops). Association for Computing Machinery, New York, NY, USA, Article 9, 1–5. DOI:https://doi.org/10.1145/2962695.2962704*

xiii Moritz Beller, Georgios Gousios, Annibale Panichella, and Andy Zaidman. 2015. When, how, and why developers (do not) test in their IDEs. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015). Association for Computing Machinery, New York, NY, USA, 179–190. DOI:https://doi.org/10.1145/2786805.2786843

---

xiv M. Beller, G. Gousios, A. Panichella, S. Proksch, S. Amann and A. Zaidman, "Developer Testing in the IDE: Patterns, Beliefs, and Behavior," in IEEE Transactions on Software Engineering, vol. 45, no. 3, pp. 261-284, 1 March 2019, doi: 10.1109/TSE.2017.2776152.

[xv] Wiklund, K., Eldh, S., Sundmark, D., & Lundqvist, K. (2017). Impediments for software test automation: A systematic literature review. *Software Testing, 27*.

[xvi] T. Dybå, E. Arisholm, D. I. K. Sjoberg, J. E. Hannay and F. Shull, "Are Two Heads Better than One? On the Effectiveness of Pair Programming," in IEEE Software, vol. 24, no. 6, pp. 12-15, Nov.-Dec. 2007, doi: 10.1109/MS.2007.158.