Robert Hill
Student number: 189047532
Student ID: rdh36

Word count: 8053

Project Genesis

A Software Development Fantasy

Project Genesis

A Software Development Fantasy

Dissertation Draft

Robert Hill

Student number: 189047532

Student ID: rdh36

Project Genesis

A Software Development Fantasy

Dissertation Draft

Robert Hill

Student number: 189047532

Student ID: rdh36

Project Genesis

A Software Development Fantasy

Dissertation Draft

Robert Hill

Student number: 189047532

Student ID: rdh36

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

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

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].

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.

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

Robert Hill
Student number: 189047532
Student ID: rdh36

- 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

Project Genesis

A Software Development Fantasy

Dissertation Draft

Research Question and Hypothesis

rolled into next section?

Robert Hill
Student number: 189047532
Student ID: rdh36

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.

Where possible, existing research has been used to determine the importance and impact of software development practices in the game.

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

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.

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 .

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.

Robert Hill
Student number: 189047532
Student ID: rdh36

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.

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.

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

Specification

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.

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

Robert Hill
Student number: 189047532
Student ID: rdh36

    - 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

Gameplay

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.

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.

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.

The importance of the 5 roles

There are 4 roles available to hire:

- Software Engineer

Robert Hill
Student number: 189047532
Student ID: rdh36

    - 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.

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.

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.

Project Genesis

A Software Development Fantasy

Dissertation Draft

Robert Hill

Student number: 189047532

Student ID: rdh36

Robert Hill
Student number: 189047532
Student ID: rdh36

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.

Software Design

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. The game is comprised of 14 scenes: BootScene, VacanciesScene, MainScene, CreditsScene, SprintBoardScene, TeamScene, HiringScene, SprintScene, SprintReviewScene, ProductBacklogScene, CustomerScene, AttributesScene, RetrospectiveScene, and EndScene.

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, and the FSM too restrictive for the iterative approach I have taken to 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 the 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 state creates an instance of the LSM to hold a sequence of states use within each Sprint.

A generic NavigationState was created and a navigationStateFactory method to simplify 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.

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.

Robert Hill
Student number: 189047532
Student ID: rdh36

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.

Robert Hill
Student number: 189047532
Student ID: rdh36

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)

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

Robert Hill
Student number: 189047532
Student ID: rdh36

- 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.

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

Robert Hill
Student number: 189047532
Student ID: rdh36

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).

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.

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.

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).

Robert Hill
Student number: 189047532
Student ID: rdh36

- 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.

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

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.

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 are 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.

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.

Robert Hill
Student number: 189047532
Student ID: rdh36

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.

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 and 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.

Robert Hill
Student number: 189047532
Student ID: rdh36

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.

Player Learning

player learning opportunities

Robert Hill
Student number: 189047532
Student ID: rdh36

Assessment

User Assessment Methodology

To measure the success of the project, test subjects will play the game and their learning responses will be estimated via a questionnaire.

The design of the questionnaire focussed on whether the test subjects felt the game was a useful learning tool and whether they learned more about the 23 software development practices described earlier and how these practices impact the success of a software development project.

Limitations

game and survey limitations

Robert Hill
Student number: 189047532
Student ID: rdh36

Results

survey results

Robert Hill
Student number: 189047532
Student ID: rdh36

Analysis

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.

Robert Hill
Student number: 189047532
Student ID: rdh36

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.

Player Survey

analysis of survey results

The Game as a Learning Tool

analysis of the game success as a learning tool

Project Genesis

A Software Development Fantasy

Dissertation Draft

Robert Hill

Student number: 189047532

Student ID: rdh36

Robert Hill
Student number: 189047532
Student ID: rdh36

Conclusion

Summary

summary

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.

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.

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.

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

Randomness

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

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.

Robert Hill
Student number: 189047532
Student ID: rdh36

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.

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.

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.

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.

Conclusion

conclusion

Robert Hill
Student number: 189047532
Student ID: rdh36

[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.

Robert Hill
Student number: 189047532
Student ID: rdh36

[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.