Software Development Fantasy

Interim Report

Robert Hill

Student number: 189047532

Student ID: rdh36

Word count: 3853

Software Development Fantasy

Interim Report

Robert Hill

Student number: 189047532

Student ID: rdh36

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 the software development process is broken 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

- 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

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 will be used to determine the importance and impact of software development practices in the game.

Anticipated Challenges

Accurately modelling the software development process well enough to simulate all the moving parts of a software project will be difficult. Understanding the relative weights and impacts of the inputs will take trial and error. To make this process manageable, I will start by modelling a single input (such as developer programming skill level), and adding additional input incrementally, and evaluating how relevant they are before continuing. Time constraints will necessarily force this process to stop, but the objective of this project is to continue until the simulation is “good enough”.

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

The choice of languages/libraries used to build the game will have an impact on the speed of development, and as a result, the realism of the finished simulation.

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.

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

Quality Assurance Practices

In *When, how, and why developers (do not) test in their IDEs*[xi] (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)[xii] the same results were demonstrated for C# developers.

Wiklund et al. (2017)[xiii] 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)[xiv] 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

Software Development Fantasy

Interim Report

Project Specification

Overview

The software development fantasy game “CTO Fantasy” will put you in the role of a CTO (chief technical officer) at a small but growing software development company, in charge of a new project, “Project Genesis”.

As CTO, it will be your role to make decisions about the technology and software development practices used at the organisation, to make changes and/or influence others to make changes. The changes you make may be explicit, such instigating a company-wide 80% code coverage threshold, or implicit, such as improving the level of software knowledge by encouraging weekly Tech Talks.

Throughout the game there will be opportunities for you to discover new practices and choose whether to try to use them, for example, learning about 3 Amigos sessions by sending a Scrum Master to an Agile conference.

The goal will be to deliver a project on time and within budget while meeting your customer’s expectations. You 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, scrum masters and product owners.

Much of the organisation state will be hidden from you, and you will need to infer how things are going by indirect means, such as the number of bugs, number of customer complaints or employee churn rate.

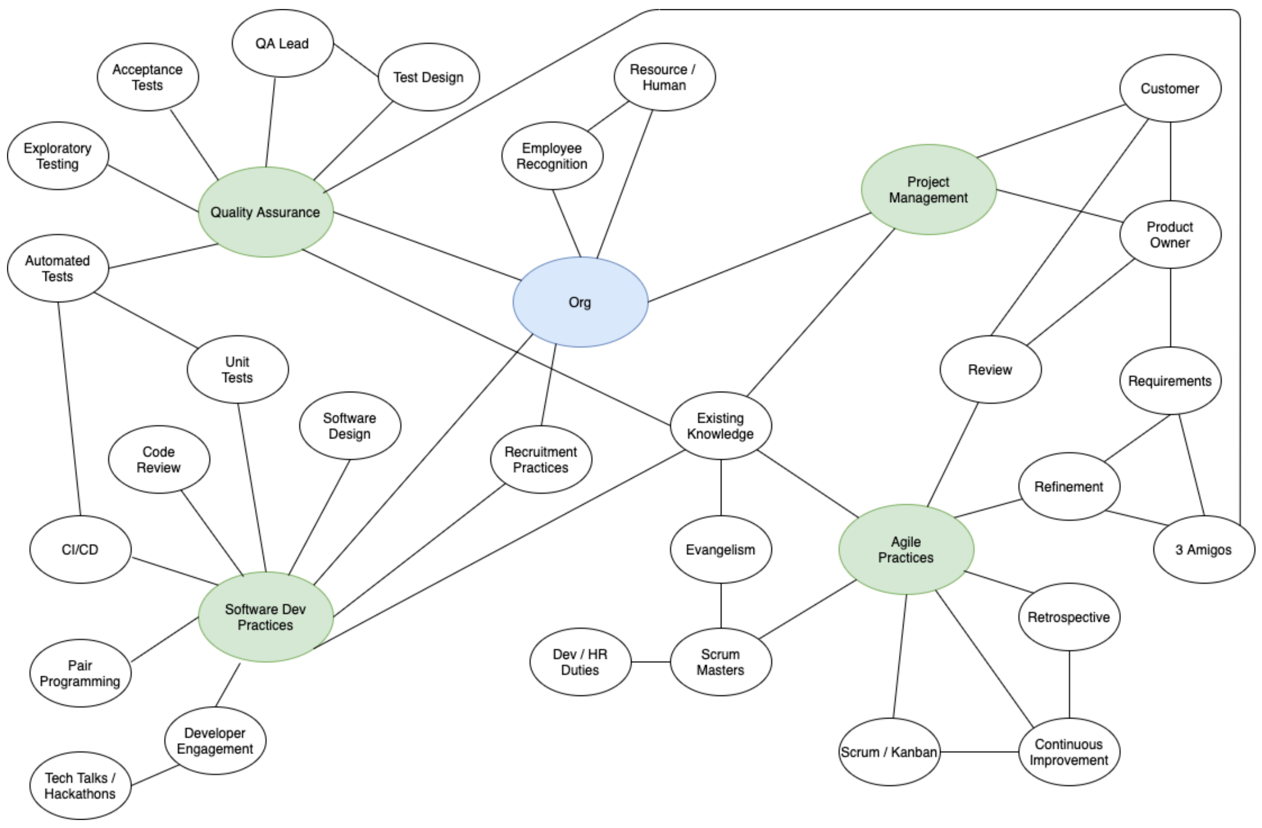
Simulation

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
  - Shift Left testing (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

An outline of some possible simulation properties and their relationships:



Gameplay

Starting Out

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

Before game rounds commence, the CTO 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 CTO can also choose to delay the start of product work in order to get some improvements in place, such as:

- Unit tests.
- CI/CD pipelines.
- Team building.
- Agile training.

Though they do so with some risk of delaying the delivery of features and a dissatisfied customer.

At the beginning of the game, the CTO must take on some of the responsibilities of a scrum master and product owner, until they are able to hire people for 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 CTO must manage the Sprint Events (Sprint Planning, Daily Scrum, Sprint Review and Sprint Retrospective), and handle any unexpected situations.

The CTO 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, happiness in the role 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 set the right goal and whether they delivered it.

At any point during the Sprint, unexpected events can happen, forcing the CTO 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 begins.

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 motivations.
- Sending team members for training or to conferences.
- Improving the company engineering culture.
- Dealing with resignations.

Mini-games

The game needs a way to know more about the knowledge of the player, and represent this as processes and practices that the CTO already knows. A series of simple mini-games will be used for this, such as matching words to concepts or multiple-choice questions. Further research into gamification will be required to decide on appropriate approaches.

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 any web-capable device, including web browsers, mobile browsers or within a mobile app.

Project Objectives

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 has not yet been determined, but it will be formed around estimating of the test subjects' knowledge of best practices in the following 3 areas has improved:

- Agile
- Quality Assurance
- Software Engineering

If it is not possible to complete a realistic simulation of the software development process in the time available, success could be measured by the level of completion in different areas.

Current Progress

Implementation

Frameworks/Libraries Used

Initially the game was constructed primarily as a React/Redux JavaScript application, with a central area built using the Phaser JavaScript game library primarily to show a visual representation of the current game state. The idea was to show an office and development team busily working away, walking between their desks and whiteboards to design software, with computers catching fire whenever they are firefighting. The idea followed similar games such as Game Dev Tycoon and Game Dev Story.

The initial plans to include animations were scaled back due to the time constraints of the project and the React/Redux parts were removed. It was felt that as a game development framework Phaser was suitable for the job.

Representation of the Simulation

The simulation and game state is modelled by collections of properties, with each property having a value between 0 and 1. The values of these properties are generated randomly, but could also be loaded from a predefined config to create a game with a known start state (potentially useful for teaching specific scenarios, but outside of the scope of this project).

Some game state properties are derived from other properties (as amalgamations of ratios, inversions or averages). For example:

- The skill of a team is derived from the mean average skill of the team's members.
- The "bugginess" of the work done in a Sprint is derived from the mean average of the inverse of the team's skill, quality mindset and collaborations (all three being derived from the members' own values).

These properties and derived properties are used (along with a little random variation) to calculate everything else in the game. For example:

- The number of bugs created during a Sprint is calculated from the "bugginess" multiplied by a BUGINESS_RATIO game config value.

Much of the remaining work involves iteratively adding more properties and derived properties until a satisfactory simulation is achieved.

Game Behavioural Model

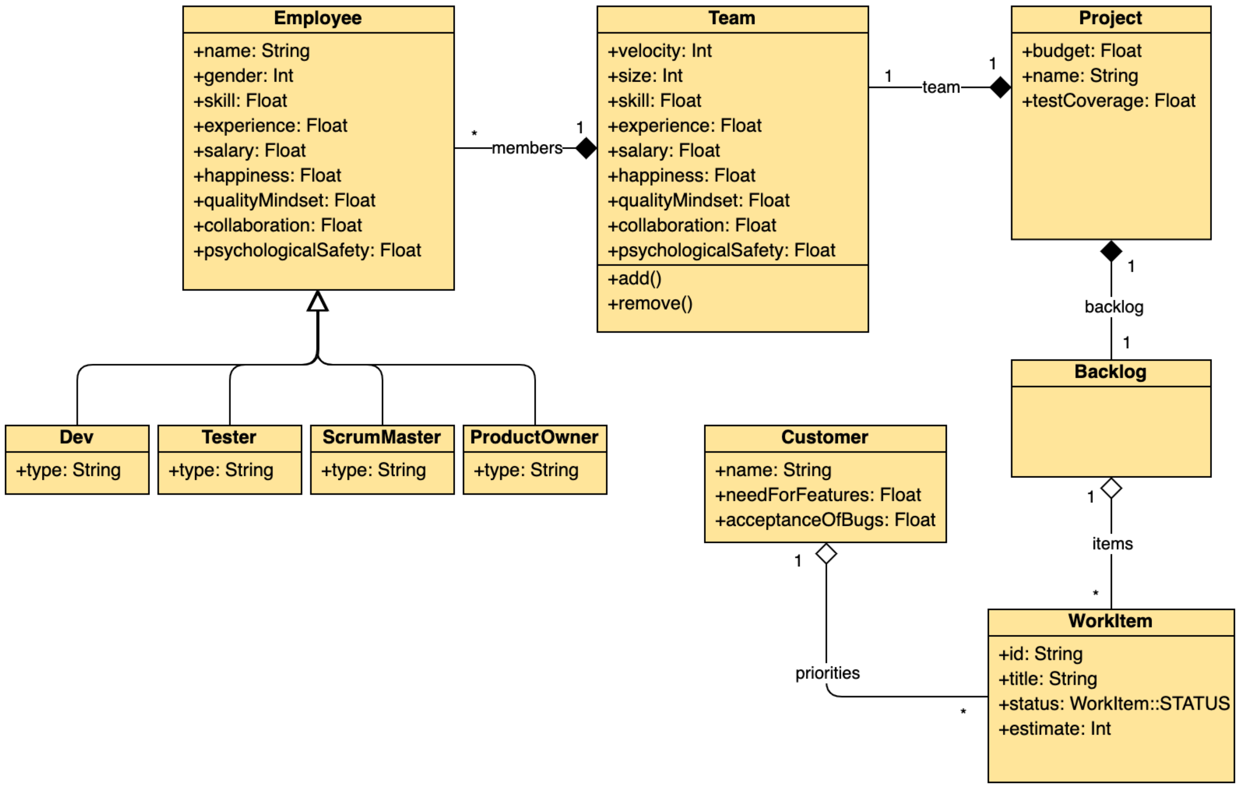
Several approaches to game state transitions were experimented with. Initially an event-based PubSub approach was used, then later a Finite State Machine. The event-based approach was a little disorganised, and the FSM too restrictive, so a simpler polymorphic Linear State Machine was created. State objects can be added to either end of a queue, run arbitrary code when executed and trigger the opening of the next state when closed. These state object are used to wrap Phaser "scenes" that load different views, and also to execute PubSub events (which have been retained to reduce dependencies between modules).

The Linear State Machine 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.

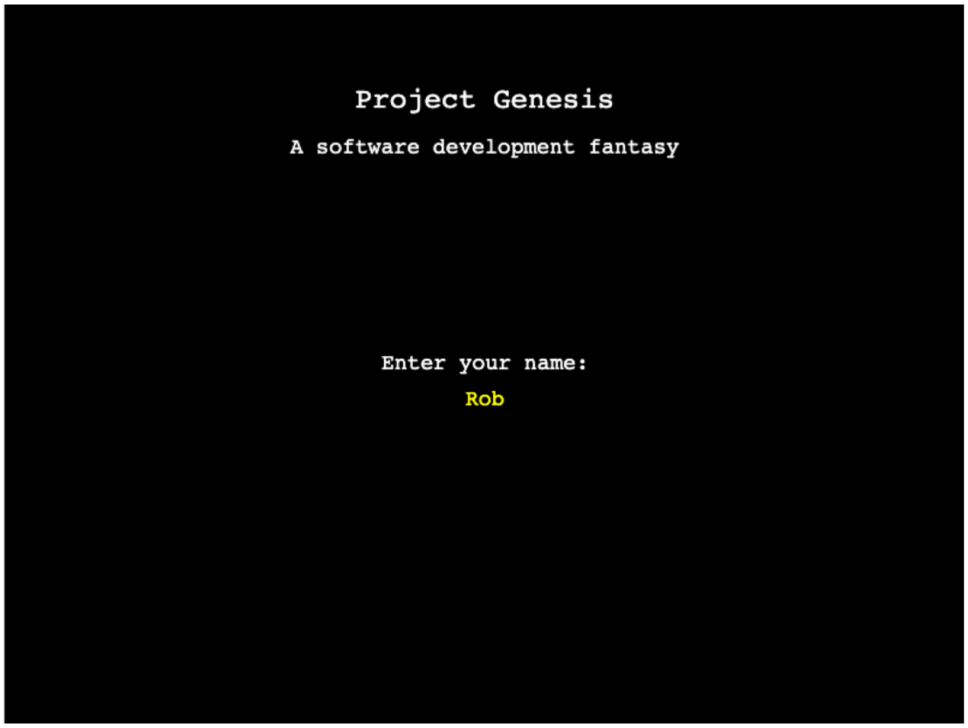
Primary Game Objects

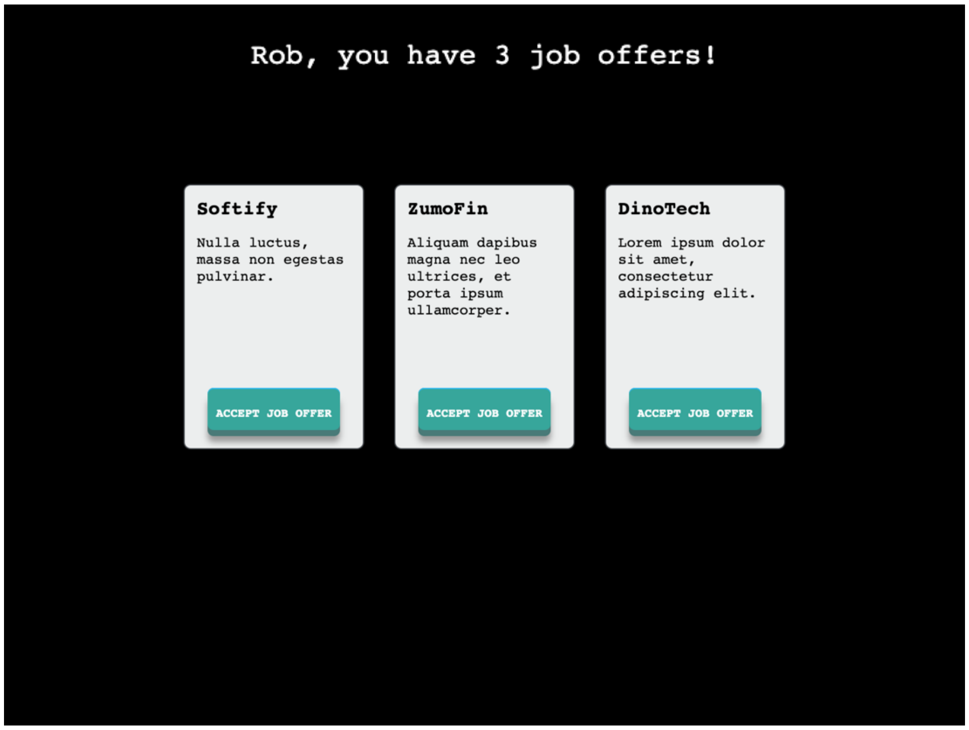
The source code contains many objects and classes who's function is to display visual elements on screen, or to navigate between different game "scenes". These are necessary artefacts of a game produced in the Phaser library.

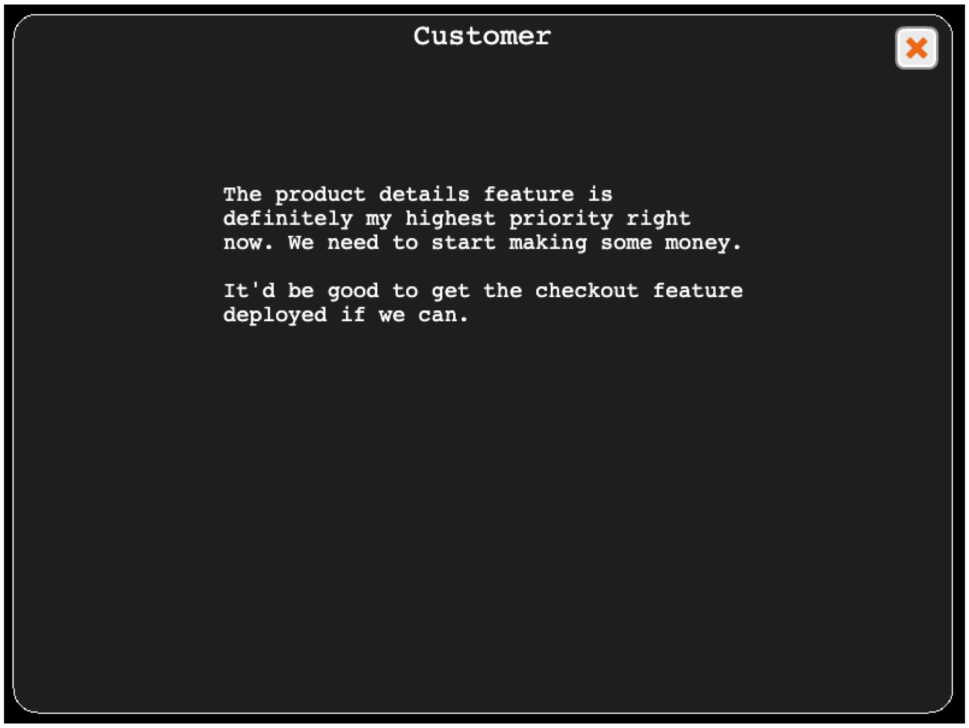
However, the simulation model is represented via the following class diagrams:



Screenshots of work in progress


Project Genesis
A software development fantasy
Enter your name:
Rob

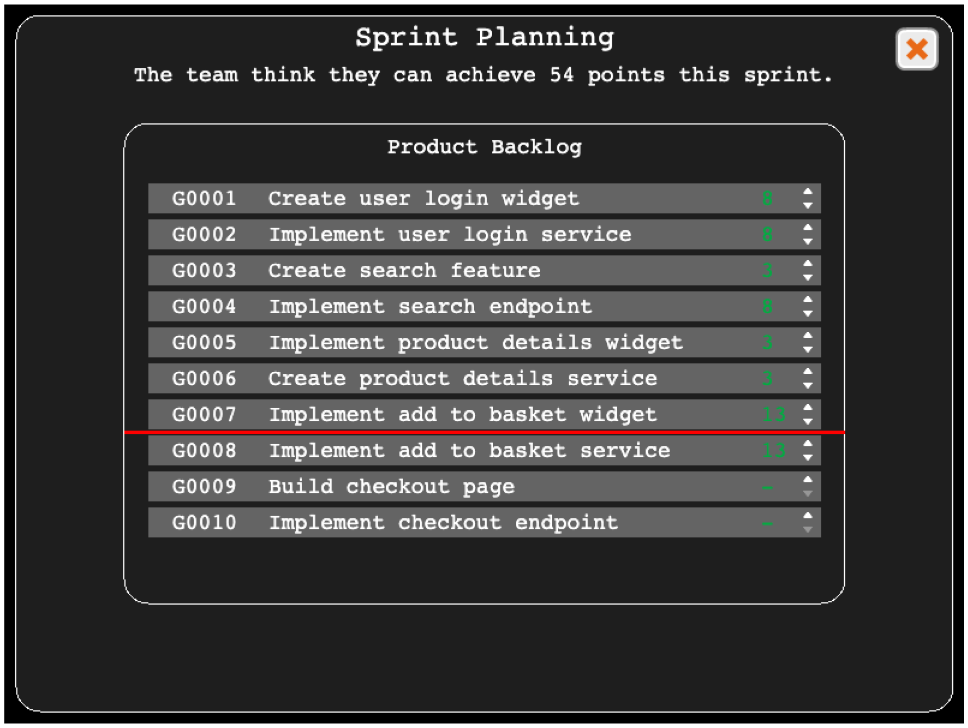

Rob, you have 3 job offers!
Softify
Nulla luctus, massa non egestas pulvinar.
ACCEPT JOB OFFER
ZumoFin
Aliquam dapibus magna nec leo ultrices, et porta ipsum ullamcorper.
ACCEPT JOB OFFER
DinoTech
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
ACCEPT JOB OFFER



Customer

The product details feature is definitely my highest priority right now. We need to start making some money.

It'd be good to get the checkout feature deployed if we can.

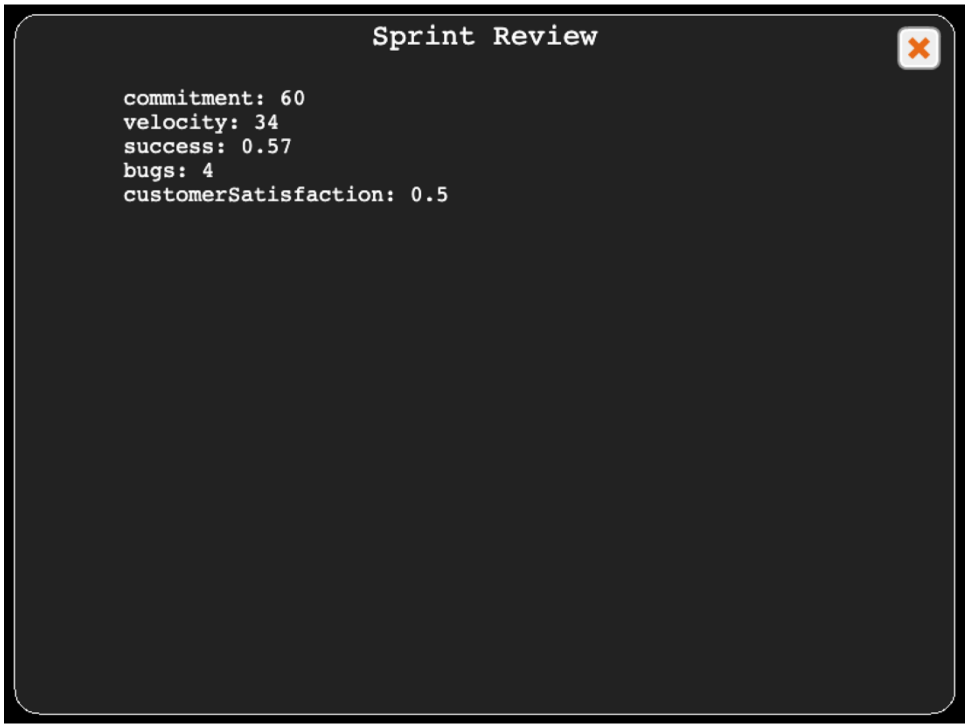


Sprint Planning

The team think they can achieve 54 points this sprint.

Product Backlog

G0001	Create user login widget	8
G0002	Implement user login service	8
G0003	Create search feature	3
G0004	Implement search endpoint	8
G0005	Implement product details widget	3
G0006	Create product details service	3
G0007	Implement add to basket widget	13
G0008	Implement add to basket service	13
G0009	Build checkout page	-
G0010	Implement checkout endpoint	-



```
commitment: 60
velocity: 34
success: 0.57
bugs: 4
customerSatisfaction: 0.5
```

Analysis

The work completed so far has concerned the mechanics of representing sprints in progress, and judging the success of those sprints by calculating work done, bugs created and customer satisfaction. These calculations depend on the various core game properties and derived properties previously discussed (skill, quality mindset, collaboration, etc.).

As yet there is little in the way of a reward system in the game other than seeing whether each sprint was a success and whether the customer is satisfied.

The reward system needs to be improved further to ensure that the game is actually fun to play. I think the following areas could provide reward mechanisms without modification to the original plan.

- Making the "on time and within budget while meeting your customer's expectations" goals visually explicit via progress bars, graphs or other means that are displayed on screen all or most of the time.
- Making the success/failure of each sprint more visually explicit.
- Making the customer's satisfaction (or lack of) more explicit at the end of each sprint.
- Making the teams happiness or levels of stress more visually explicit.
- Making negatives more explicit, such as new bugs, or low code quality.
- Introducing firefighting events.

I think these reward systems will be enough. However, an alternative would be to introduce direct competition via a scoreboard, so that players could compete against each other for better scores and to improve their own scores by playing again and making different decisions.

I already had some familiarity with the Phaser game framework, so anticipated that it would be technically straightforward to use. However, I had forgotten quite a lot and the framework has changed significantly over the last few years. I faced some challenges just getting game object to display correctly on screen. As I have continued to use Phaser, this issue has lessened.

My greatest challenge so far has been identifying existing research that backs up the core principles of the game and what it is attempting to teach. I've been able to find very limited research on what constitutes good or bad Agile practices, and what the tangible benefits are of well-executed Agile. To some degree this is also true of the other 2 topics, Quality Assurance and Software Engineering. I will continue my literature search, but existing reviews of the literature seem to indicate that this is an issue with the available research.

Dissertation Outline

- Abstract
- Introduction
  - The Problem
  - Research Question and Hypothesis
- Literature Review
- Implementation
  - Software development as a simulation
  - Gamification mechanisms
  - Representing player knowledge
  - Selecting and introducing learning events
  - Responding to player actions/choices
  - Determining success
- Assessment
  - User assessment methodology
  - Limitations
- Results
- Conclusion
  - Summary
  - Future Work
  - Conclusion

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 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

xii 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.

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