University of Glasgow | School of Computing Science

Level 3 Project Case Study Dissertation

AMIS - A Marketplace for Industrial Symbiosis

Chris Brown
Alex Smith
Timothy Ness
Paulius Jonusas
Ross Wilson

3 April 2016

Abstract

Technology has the potential to facilitate solutions to the challenges of supply and demand. Sponsored by the Crichton Carbon Centre (CCC), whose aim is to explore the challenges and promote the benefits of a low carbon society, the AMIS (A Marketplace for Industrial Symbiosis) project is inspired by the old adage 'one persons garbage is another persons treasure'. Recognising that the commercial use of landfill has failed to reduce at the same sort of rate as domestic landfill use, an auction site for business waste was developed in collaboration with staff at the CCC. The auction site creates an effective platform for businesses producing waste to match with businesses who can use the waste, perpetuating a circular economy. In particular, Small and Medium Enterprises (SMEs) can view and bid for products. This has the effect of stimulating a demand for waste items where none previously existed. The website was developed using Ruby on Rails and the project was managed using an Agile methodology. The deliverables included login, trading posts, submitting bids, securing deals and payment through PayPal.

Education Use Consent

1 Introduction

Case study is a powerful empirical method which is used to construct theories or understand and explain phenomena. It is of particular assistance in software engineering when exploring and developing an understanding of the capabilities of new technologies, processes and organisational structures.

The University of Glasgow's Professional Software Development (PSD3) course has covered modern software development methods, agile methodology, and various techniques and good practices for both developing code and working with customers. Through this particular case study we as a team hoped to gain a greater understanding of why and how these skills are applied to a large-scale project.

This paper presents a case study of the software development process we went through when building an online Marketplace for Industrial Symbiosis (AMIS). AMIS is a proof-of-concept project, based on Rails, a popular web framework for Ruby, which aims to facilitate the trading between SMEs of by-products and industrial waste. The scheme would initially be available only to Scottish companies but it has the potential to be extended far beyond just Scotland. The goal of the project is to encourage businesses to focus on the reuse and recycle aspects of waste hierarchy, bringing multiple ecological and financial benefits to the participants. This is particularly true in current times as recycling costs and landfill taxes rise. As an online marketplace, AMIS is easy to access, and this is an important aspect of the scheme. The project has the potential to expand into other fields in the future, such as trading in spare haulage capacity, which could be considered a species of waste in its own right.

The case study is structured as follows. Section 2 presents the background of the case study in more depth, describing the customer and project context, aims and objectives and the project state at the time of writing. Section 3 reflects on the software development process undertaken throughout the project. It is divided into six subsections. The first subsection 3.1 describes the main framework chosen for the project and our reasoning for this decision. In the second section 3.2, we will reflect on this and the other initial design decisions we made, describing the main issues we encountered. Each of the subsequent sections 3.3, 3.4, 3.5 and 3.6 reflect on one of the four iterations we underwent during the project. Each overviews the progress we made and discusses the challenges we faced during that iteration e.g. learning and working with new and varied technologies. In particular, we will evaluate how we worked together as a team to ensure good communication and task management. We will also look at our software development practices, such as test-driven development, use of version control and continuous integration systems at each stage of the software development process.

2 Case Study Background

The Crichton Carbon Centre (CCC) is an organisation based in Dumfries and Galloway which is committed to helping reduce carbon emissions in that region and across Scotland. The organisation was founded in 2007 and since then has been trying to drive a change in Scotland towards a more circular economy. A circular economy is an economy in which the waste products from one industrial company, rather than being scrapped, are sold to another company to be used in its production line, with the aim of reducing waste and removing the costs of disposing of waste for both businesses.

Throughout the project we were interacting with two employees of the CCC. Primarily we were interacting with Dr Liz Brooks, a post-doctoral Research Fellow at the CCC and the Crichton Institute. She attended each iteration meeting and was our main point of contact throughout the project. We also engaged with Professor Mike Bonaventura, CEO of the CCC. He attended both the initial requirements elicitation meeting and the final iteration meeting. His partial technical knowledge proved useful when discussing some of the technical design decisions we were making.

The motivation for the project was as follows: domestic waste recycling (which is handled by local authorities) is now well established in Scotland and has been successfully reducing the amount of waste going to landfill. According to the Scottish Environment Protection Agency (SEPA), "[f]or the 2014 calendar year, the household waste recycling rate using a new calculation methodology was 42.8%" [?]. This means there has been a large decrease in the amount of household waste being landfilled: "[t]he quantity of household waste landfilled has fallen by 16.6% since 2011" [?]. However, the recycling of waste from businesses in Scotland has been lagging behind. To quote the project specification as provided on Moodle: "In 2010 in Scotland alone, one third of the 8.3 million tonnes of the waste generated by commerce and industry went to landfill" [?]. This is an unsustainable rate and if it continues will have severe consequences for the environment. A large proportion of the materials going to landfill still have value but companies are often either unaware of the market for their waste goods or are unable to find buyers. Businesses are also unnecessarily paying a huge cost to dispose of their waste: "[i]n 2008 the manufacturing sector produced 22.7 million tonnes of waste, at a cost of 1.8Bn at prevailing rates of recycling costs and landfill taxes" [?]. With constant changes in EU directives these costs are only going to rise: "[t]he cost to Scottish business of waste going to landfill in 2010 was 220M, which will rise to 50% by 2020" [?]. This additional cost will negatively impact business across Scotland, especially in the future as raw materials become scarcer and therefore increase in value.

The Sustainable Process Improvement (SPI) project (begun in April 2012), worked with businesses in Dumfries and Galloway and across the south of Scotland with the aim to "identify opportunities for implementing or improving resource efficiency business practice" [?]. When the project terminated in April 2015, many of the participating businesses expressed further interest in improving their resource efficiency and so the AMIS (A Marketplace for Industrial Symbiosis) project was born.

The idea behind the AMIS project is to create an online marketplace for Small and Medium sized Enterprises (SMEs), particularly those in industries with high quantities of waste and/or high costs associated with disposing of waste, to buy and sell waste products. Currently it is difficult for the roughly 39,000 SMEs [?] to find buyers for their waste but the aim of the online marketplace is to connect people, who would otherwise be unaware of each other's needs and the going price of such goods, in order that they can trade waste products and prevent the waste products from going to landfill. The need to sell waste products is becoming more urgent due to new, more stricter EU legislation on waste disposal. Purchasing waste products suitable for their needs rather than buying raw materials directly from source may also enable SMEs to save considerable amounts of money. By incorporating these waste products into manufacturing processes, businesses will also be helping the industrial sector to focus on the 'reuse and recycle' stages of the waste hierarchy. An extension of this idea is to incorporate logistics into the circular economy notion. According to the Road Freight Statistics 2014 report published by the Department for Transport, 29% of lorries on Britain's roads are running empty, that is "carrying zero tonnes for [the] whole journey from origin to destination" [?]. Predictable spare capacity in freight vehicles could be advertised and sold off to SMEs.

Before attempting to resolve the issues outlined, we had to consider the possible reasons for the lack of such a solution to date. These reasons included the difficulties in designing an online marketplace which would be useful and readily used by SMEs. In order for the site to function there would need to be a critical mass of users such that a buyer will readily be able to find what they are looking for and a seller will be able to find a buyer in a time period that is practical. The online marketplace would therefore need to cater to the varying needs of different types of business to attract the necessary number of users. Further difficulties arise as a result of the complex rules regulating the trading of industrial waste and by-products. Strict legislation prevents the trading of both illegitimately obtained and potentially hazardous waste. Each of these difficulties needed to be considered in any outlined solution.

Our project brief was to design and implement a fully functional 'proof of concept' for the proposed online AMIS marketplace. During the project we met with the representatives of the CCC five times to elicit the requirements of the AMIS marketplace and to demonstrate our proposed solutions. In doing so, the following requirements were established: the implemented web application would include a registration process to provide security with some additional privileges for only administrative users, methods to assure users are buying and selling only legitimate goods and are obeying all legislation, different types of purchases (buy-it-now, auctions, subscriptions/ long term deals and split sales), PayPal integration to support the secure payment for items, a statistics page to emphasise the positive impact of the site, searching/ categorising facilities to help business representatives find what they were looking for, a feedback system to show the reliability of sellers, methods to protect the sellers information if they are concerned about the sensitivity of their products and a full record of all completed trades that can be used to provide evidence of the legitimacy of the trading on the application. All of these features were to be implemented in a user-friendly interface and have the necessary information supplied to explain their use and purpose. The purpose of the site as a whole also had to be clear. In addition, we were to propose a cost model for the application including projected costs incurred, such as through servers and administration staff, as well as explaining how the site could earn money.

The scope of potential requirements is huge and therefore far beyond what we could have feasibly implemented in the duration of this project. As a result the requirements were considered using the MoSCoW system for deciding their priority in our implementation. That is breaking the requirements into must have, should have, could have and won't have. There were lots of features that ideally a complete application in this problem domain would have but we were not able to implement them in the scope of the Team Design Project 3. These will be outlined later in this report. However what follows is a description of the core features that we did implement.

A new user of AMIS can access information about the purpose of the application and how to use it via the carousel on the home page, or by visiting the 'about us' page. A user can also view the statistics page, which has several graphs containing information about completed trades showing the environmental benefit of the application. A new user can browse the items being sold without registering or logging in. The most viewed items are visible on the home page. Other items can be found by using the search bar or using the category menu.

Users who decide they would like to use the website can register by filling in the minimal registration form. Further details, such as a profile picture and company description, can be provided on the profile page. In order to buy or sell an item traders must be approved to buy or sell an item in that category. To become approved they must select the category they want to become approved in via the profile page, and provide a PDF giving evidence of their legitimacy to buy or sell in that category. This evidence can then be viewed by an admin user who can then accept or reject the application. If successful, the trader can then buy or sell in that category.

A trader can create a new post in any of their approved categories via the profile page. The post can be chosen to be either an auction or a buy-it-now sale. If it is an auction then approved buyers have seven days to bid on the item after which the highest bidder is required to pay for the item. The highest bidder's account is frozen until the outstanding payment has been made. When creating a post, sellers can choose to prevent viewers from being able to see their location and name. They can also specify that they will deliver the product themselves, otherwise delivery will be provided by AMIS at an additional cost to the buyer. Outstanding deliveries will appear on an admin only viewable page. Sellers can also choose whether or not to provide samples of their product. An approved prospective buyer can then request a sample which will show up on the seller's profile page.

If a user suspects that a post is not obeying the protocol of the marketplace they can then use a button on the post view page to report it. All reported posts will be reviewed by an admin user who can either dismiss it or remove the post if it is indeed illegitimate.

The web application includes integration with PayPal to provide a secure method for payment. This proved a complex feature to integrate and has therefore only been implemented in a limited form. All payments are made to one PayPal account. This could be extended in the future such that payments are made directly into the seller's account.

The profile page contains a complete list of a traders current posts, sold items, purchased items, bids made and samples requested of them. Once a trade has been completed a buyer can then give feedback on the trade. Feedback is given in two ways: a star rating (out of 5) which is averaged with all other feedback to give an overall rating for a trader, and a textual comment which is attached to the completed trade. An admin user can view all the completed trades in the last 7 days and can download a spreadsheet of all historic trades. An example tour of the website has also been implemented. This could be extended to be an effective way of explaining to a new user how to use the application.

3 Reflection

3.1 Justification of Design Decisions

An extremely important design decision we had to make at an early stage of the project was determining which technology we would use to produce the software. The chosen software platform would have to provide all the necessary functionality whilst making the development process as straightforward as possible. Since the project was fundamentally to develop a web application, we had a wide range of web application frameworks to choose from. We quickly narrowed this down to two: Ruby on Rails and Django. The reason for this was because, as a team, we had experience in both these frameworks. Everyone in the group had used Django before but only one person had used Ruby on Rails. After thoroughly researching both technologies we came to the conclusion that Rails was more suited to our web application. Despite the inexperience of several members of the team, we decided that the benefits of the framework outweighed the initial learning curve. Some of the reasons for this decision are outlined below.

A big selling point of Rails is the speed at which you can get a basic web application up and running. This, however, is at the expense of control over the fine details of the application. However, the purpose of the application, as a 'proof of concept', was not to create a finely polished completed product but to demonstrate as many working features as possible that a fully functioning application would have. This sort of development would be very suited to Rails.

Part of the reason for the fast development of Rails applications is the rich array of 'Gems' available. Gems are third party, open source, pieces of software which can be assimilated into a project to add customisable functionality quickly. The development process of an application therefore becomes less about writing code from scratch and more about bringing a collection of these gems together and using them to create the required functionality. This illustrates well the software engineering principle of code re-use which allows code to be created quickly and correctly (these gems are often very extensive and thoroughly tested and as such are less likely to contain errors than code we have written from scratch). We used Gems in areas such as testing, user authentication, image handling and front end formatting. Although Django has plug-ins for a similar purpose, the Rails mechanism for Gems makes it incredible easy to find and install them. Most Gems also have extensive and well-written documentation which enabled us to determine how they could best be incorporated into our project.

A persuasive motivation for choosing Rails was the substantial security requirements of our project. From the beginning the client made it clear that security was an important issue for the application. The project specification explained that the site needed to be secure in at least three ways, those being: user authentication and secure sign on, roles and access privileges of users and secure transactions and external linkages. In order to add user authentication to the application easily we used a popular Gem called Devise. Devise is an implementation of user authentication and authorisation which permitted us to control a large part of the user model in our application. It provides functionality to create a user given the appropriate fields, check email and password validity and to encrypt passwords for safe storage. We also used its implementation of forgotten password recovery and password reset. This Gem was very valuable to us because it saved us a lot of time during the project implementation, as well as establishing strong security from the outset. Devise is a widely used Gem which means that it has already been extensively tested and improved far beyond what we could have achieved with an implementation we had written from scratch. We were able to extend the devise user model to include an admin field, allowing us to differentiate between a regular user and a user with administrative privileges. This provided security for the content needing to be kept private.

Finally, in order to satisfy the requirement that transactions be carried out securely, we chose to add PayPal functionality into the application. Interacting with PayPal turned out to be a hugely complex process (the PayPal interface specification is difficult to understand and use). Fortunately, Rails has an SDK (software development kit) Gem for facilitating interactions between the application and PayPal. This greatly simplified the process as it no longer required interacting with the PayPal interface directly but using the functions contained within the Gem. The Gem code included examples of how to use it to carry out various kinds of PayPal transactions which were invaluable to us as we had no previous experience of developing PayPal integration.

Testing is crucial to developing successful code and therefore was important to consider when making a choice of which framework to use. Thanks to Rails' built-in automated-test-generation a large number of test cases were created allowing us to have a very wide test coverage. Since we did not have to spend time writing basic test cases we could devote our time writing more complex and project-specific tests. Along with the built-in test-generation provided by Rails itself, there is a wide range of Gems that can be incorporated. One of the Gems which we chose to include in our project was Rspec. Rspec allowed us to write specification test cases in a way which corresponded syntactically with plain English. This proved to be a real advantage to us, especially when one member of the team was developing on top of code written by another member of the team - with these clear, readable tests we were able to quickly find the sources of any errors, and address them without having to look through the code.

3.2 Reflection on Design Decisions

The development process began with the team learning how to use Rails. Most of our team had no experience using Ruby or the Ruby on Rails framework. This meant we had to set aside time to learn a new technology before we could actually start coding the application. We had therefore been working on the project for quite a while before we started developing anything which would feature in our finished project. Rails is a fairly popular tool and so there are many good online tutorials which helped us learn how to use it quickly and easily.

The next issue to arise was that there were discrepancies between the versions of both Ruby itself and the number of different Gems which Rails uses, which we each had installed and set up on our personal machines. In order for our code to work coherently we had to ensure that everyone was using the same version of each piece of software. Each Rails application contains a file called Gemfile which declares the Gems that are used in the application. Within this file each Gem has a version number associated with it. These version numbers specify the version of the Gem that is to be installed when the install command is given. Once installed on one system, the exact versions that were installed are stored in a file called Gemfile.lock. When the project is loaded on another machine, this file is inspected so that the exact same versions will be installed on that machine. Therefore using this system we were all guaranteed to be working with the same versions of the software.

The most difficult issue we had with using Ruby on Rails was getting our application to compile on the School of Computer Science server 'Hoved' and hence make use of the contiguous integration tool Jenkins. Once the administrators of the Hoved system had installed the version of Rails that we needed and created the required databases we still had great difficulty getting our application to work on the system. In order to install the Gems on the system, we required super-user privileges that we did not have. This proved extremely impractical as throughout the project we were regularly adding more and more Gems and could not repeatedly get the administrators to run the install command. We later discovered that it was possible to install the Gems locally within the project directory, which did not require super-user access. Initially we had committed these files to our repository but this was cluttering up the repository, as well as preventing the files from being correctly installed on different systems. Once these files were removed we were able to install the Gems locally in the project directory on Hoved. However, our problems were still not solved as in order to get the required commands to work we had to run them in the context of the project (adding bundle exec before any entered commands), a fact that we did not initially realise. Also, the Hoved system required a different database socket than the other systems we tested the application on. We worked around this by including a line in our build script which changed the socket specified in the database initialisation file. Eventually, after a great deal of time and effort, we did get our project to build on Jenkins. The build script included setting up and populating the database, executing the test cases and running the server. The main consequence of our code not compiling on the Hoved server for much of the project was that we were unable to use the Jenkins tool in our project to ensure that the commission of new revisions were correct until the issue was solved. In reflection, the expertise of the helping technical staff should have been considered when choosing a framework. Although they were indeed very knowledegable, they had limited experience with Ruby on Rails and so were unable to help us with some of our technical issues.

Another design decision we had to make was to choose which database to use. We wanted the database to be scalable but also easy for us to use. We settled on using MySQL because we all had experience using it and Rails has good built-in support for integrating a MySQL database into an application. Further, we knew that there would be no issues as the site grew because MySQL supports very large databases. Although this is only a proof-of-concept we wanted it to be easy to expand on without having to change much of the pre-existing structure.

We selected the Bootstrap Gem to format the front-end user interface of the application. This was an obvious choice for us because we had all used it previously and it is a very popular tool. To go along with this we used Jquery for front-end Javascript features allowing us to create a more aesthetically pleasing user interface.

3.3 Reflection on Iteration One

Early on in the project it was recognised that team members reacted well to clearly defined milestones. This was due to having somewhere for everyone to aim for and this goal setting within timelines helped in sticking to deadlines. This approach was adopted throughout the project and further improved on to set more specific tickets in Trac so each ticket was independently clear of the work it required. These milestones were also all completed which is a great habit to be in in order to deliver a good product within deadlines. The meeting was also reflected upon during the first retrospective. It was clear during the customer meeting that the team was on the same page as the customer regarding aims/goals of the project. This was good to establish early so that proceeding talks with the customers could be efficient in terms of the topics discussed. This ensured that the team were able to make the most of the meetings.

Initially there was a bit of a learning curve with Trac which was apparent in the first retrospective. Many team members found it difficult to visually check the project's progress on Trac. This did improve once the system was better understood and people became more vigilant in their use of Trac. After a couple of instances when several team members started to work on the same ticket at once we as a team sought to improve our communication skills as well as become more methodical in each step when starting a ticket in Trac. Everyone made sure to take ownership of tickets and made sure the status of the ticket was updated as it was being developed, tested and reintegrated to the trunk. Through this we saw a substantial improvement in the efficiency of our team work.

There were several reasons, already outlined in this report, why Ruby on Rails was chosen to develop the application. However, working with a new technology was not without its problems as was discovered during the first retrospective. With the new technology, members found that sometimes tasks took longer than anticipated at the poker planning meeting due to a new, unfamiliar language. This caused a change in how we estimated the costs of tasks in subsequent meetings in order to reflect this. It was clear points were slightly under inflated in the first iteration and were adjusted in later poker planning meetings and thereafter accuracy improved.

In order to mitigate the negatives of working with a new technology each team member worked through a Rails tutorial to gain some familiarity with the framework which saved a lot of time in the long run. This was useful to gain a better understanding of the high and low level of how to best use Rails which allowed team members to be able to input into design decisions taken at different stages of the development process.

Another issue facing the team due to the choice of Rails were the difficulties faced when trying to build the application on the university computers and getting a correct Jenkins build. A lack of Jenkins build meant that it was difficult to track down a broken build when every build is broken! This caused some confusion for some team members to build the application on their own PCs. It was clear once the Jenkins build was fixed what a great asset it is. Giving confidence in correctness when making changes to trunk and ensuring re-integration was not detrimental to existing features.

It was spotted that the team could be working more efficiently. Sometimes multiple people would be working on one task when one person could have been sufficient and they could then seek further help when required. Moving forward, team members tried to work towards this independent work to be more efficient but seeking help when they were really stuck. In particular, this approach was more common at the beginning of the project when people were working with an unfamiliar technology and members became more independent towards the end of the project. This change helped the team to work more efficiently in later retrospectives once members had a more fundamental grasp of Rails and developing code as part of a group and the difficulties that can be associated with it.

One of the major grievances that came up at the first iteration was the fact that connecting to SVN and committing was a bottleneck when developing. This was largely due to fact that every team member was working on personal machines. It was decided at the first iteration that it would be beneficial to create a login script for signing into the university servers. It was clear that tasks that can be automated should be. As long as the short term effort is likely to save time over the lifespan and during the development of the project and the truth was found in this. Automated scripts can also be more reliable.

At the end of the first retrospective it was clear that there was room for improvement for SVN commit messages. The first issue that is raised where commit messages have to be used in debugging the issue helps you to see practically how useful they can be. This insight can also teach what is it that makes a good commit message? which is an important question and being aware that at some stage this message could be incredibly useful when looking back months later. This seems to be an ongoing process with experience being the biggest thing for useful commit messages but after this retrospective more care was placed in giving the messages more meaning.

At the first meeting the clients explained that there would be some legal constraints as to what can/cant be sold and how some materials required documentation/licensing to sell. The clients advised us to focus on a more specific market as a way to prototype the application where it could be extended further at a later date.

Going forward there were a couple key points to improve the development process. The major one was team members making a concentrated effort to assign responsibility for their ticket/task to themselves to avoid conflicting effort. Each member also had to further improve their skills and knowledge of Rails to improve development understanding and efficiency.In keeping with the QA process team members were going to try and get more familiar with branching in SVN and reap all the benefits that can come with developing on independent branches. These benefits were realised later in the process.

3.4 Reflection on Iteration Two

Iteration two was a less intense iteration since all of the team members were preparing for December exams but there were some points that still needed to be addressed that came up at the second retrospective.

Throughout this iteration we continued to work through the issues which were stopping our project from running on the School of Computing Science server, Hoved. We found several installations were still required for the Ruby application to run which was causing the build to always fail: a major issue in pinpointing how, and when, the application's build actually failed. We continued to work on this issue however it took a lot of time and effort to resolve.

During the second iteration and demo the customer was pleased to see tests that were passing and also tests that were failing which showed the teams design methodology of evaluating TDD (Test Driven Development). Failing tests at this point were largely features that were yet to be implemented and showing functionality that would be there at a later point. However, there was a need for more tests to check user authentication to make sure the site was secure. A major issue recognised at this stage regarding security was that users could alter other user's information through manipulation of urls. To solve thid we added a ticket on Trac representing the task of thoroughly checking the URLs and authorisation required to view them. Tests to check these could have been added earlier as site security is an increasingly important challenge especially for this type of application.

During this iteration the overall design of the user interface was discussed thoroughly, in particular the specific look and feel of the site. The development of prototypes was a means to provide the client with a rough idea of potential designs and one was finally settled on. In hindsight there could have been more diverse prototyping to strengthen the choices for this part of the design. We mainly used rough sketches as prototypes of the user interface and perhaps these did not give the clients a clear enough vision of how potential users would interact with the site. We learned from this the difficulty of clearly expressing ideas and concepts to the clients. Exploring this further could have prompted more fruitful discussion on design with the customer. While this was one of the more quiet iterations there were still points to be taken on and things to learn from. Narrowing down the Jenkins build issue was good but effort would still be needed to get it fully working, which happened in a later iteration.

3.5 Reflection on Iteration Three

The third iteration of the team design project saw us spend much more time developing the implementation itself, and with the Christmas holidays falling over this time we were working individually far more than in previous iterations. This change tested our team work in new ways, including our use of Trac to delegate the implementation of certain functionality; our use of Subversion to maintain a clean and working copy of our project as we were developing simultaneously; and our communication skills despite us all being far from Glasgow. This time of agile, rapid application development uncovered strengths as well as weaknesses to learn from and grow in both for us as a team and each of us as software developers.

After deciding to change the style of our tickets on Trac from each ticket being derived from a single user story where numerous overlapped and some were large and abstract, to each being a very specific and well defined feature, it became far easier for each of us to know what needed to be worked on. This change saw a much fairer separation of tasks, no longer was a single block of functionality being left to one person alone, and the whole application grew more gradually with each small feature being individually examined and tested before being committed far more thoroughly than it would have if developed as part of a larger block of features. For each of us as individual developers it was considerably easier to work on these tickets having spent the time as a group defining them because it was far clearer what the feature was for and what was required of it. Another obvious benefit of this new format of Trac tickets was the separation of concerns for each feature and increase in cohesion of the overall system which made future modifications to the code far less complex and reduced the risk of them affecting other features.

There were some definite challenges with this new ticket style: primarily the assignment of tickets to developers was at first unclear and led to some features being started by two different team members at the same time. We believe that this issue started during this stage of the development process because of the increase in the sheer number of features being implemented but also because of the new ticket style. With higher cohesion the tickets took less time to complete and lower coupling led to many features being related by a more abstract overlying function of the system. This made it easy for each of us to develop for one or more ticket without changing the ownership field on Trac or consulting the rest of the team. From this we learned that the efficiency of our project's development is reliant on the conglomeration of our schedules and what we have resolved to work on as well as our ideas for the project itself.

Good use of a version control system is imperative to the success of team-based software development; as a team we were especially aware of its importance during this time of rapid application development. During the third iteration we became aware that we were not making sufficient use of subversions branches. A consequence of us all developing often large features on the single trunk simultaneously was complicated merge conflicts, and on a couple of occasions features ceased to work correctly after changes were made to a method or model on which they relied. At our retrospective we discussed how these issues could largely be avoided by developing features in branches, separate from the global copy, and merged only when there are no conflicts.

In the paper “The Impact of Tangled Code Change” [?] an investigation of five Java open source projects revealed 15% of all bug fixes consisted of multiple tangled changes - the commission of multiple unrelated code changes together. We also discovered this problem within our project, with some commits containing multiple unrelated changes and creating some confusion as to what a revision actually contained. Our solution to this issue was as simple as committing more frequently because all features and bugs were already implicitly defined on Trac, solidifying the intrinsic direction of the project.

One distinct drawback of our team’s system for contiguous integration during the third iteration was the fact we could not yet run our application on Hoved and therefore could not make use of the contiguous integration tool Jenkins. Without this tool there were a few occasions when the commission of a revision caused the global copy of our project, or features of it, to break without us knowing. Noting these issues we sought to rectify this problem, which we went on to do by the forth iteration.

However during this time we were forced to establish strong communication across the team which was invaluable not only for solving the issues which we had but also for introducing new ideas, revealing inconsistencies with the design and for getting feedback from each other on implemented features immediately. An instance when we saw how helpful such communication was to the project was when developing the user interface. Being a fairly complex, functionally diverse site, intended for users with a wide range of experience with technology, a well designed user interface is critical to the usability of the application. Being able to test the user interface simply by having each other use it was significant and lead to the user interface becoming stronger and more usable very quickly.

A strength of our team during this iteration was communication and an aspect of this which we all worked on during the iteration was writing meaningful commit messages to help each other know all changes which had been made since they last updated their local version. This was something which we discussed at the previous retrospective meeting and resolved to change. This small change had a surprisingly positive effect on the development process: it helped us discover the cause of large errors more quickly despite not having the Jenkins tool up and working; it acted as a summary of the application’s state of implementation; and it also ensured each of us thought through all of the work which we had just done.

As the iteration continued we also discovered that we, as a team, needed to improve the commenting of our code. This was first exposed when trying to write additional test cases for the models to enforce security issues which we discussed with the customer in the second iteration meeting. The amount of time these took to write was longer than estimated because it was unclear what some of the methods within the models, written to enforce security issues, were supposed to do. As the iteration went on this became an issue elsewhere, including within the controllers when extending the functionality within pages and within the HTML rendering the pages themselves. Although this did not cause any errors within the application it slowed down the development slightly. As a team we resolved to write meaningful comments describing new methods being written and went through the existing code adding similar comments, to make the application more maintainable.

Overall the third iteration was one of the most challenging since we were trying to implement as much of the functionality as possible within a relatively short time to test our design decisions early. As a result it saw the software evolve more rapidly than in any other iteration. Additionally, as a team it required of us expeditious development of teamwork which pushed us as individuals to lean into the project and its holistic goal to create an online market for industrial symbiosis.

3.6 Reflection on Iteration Four

Upon coming to the final iteration our team was operating far more smoothly than on the offset of the project, implementing new features was considerably faster; modifications to existing code more simple; and the comprehensive system of project development, noticeably more agile. Practically, we worked on the smaller features which were not key to the functionality of the application, as well as developing the system's non-functional requirements and polishing the functional requirements after they were inspected alongside the customer. However, as a team we continued learning from the development process: strengths including using unit tests to ensure quality as well as effectively delegating and collaborating on tasks; and weaknesses including writing helpful documentation and the use of database migrations.

This iteration was the first full iteration in which we could use the contiguous integration tool Jenkins, since up until this point the application was not able to be run on Hoved. The benefit of this tool was seen almost immediately: we saw a couple of instances when revisions being committed caused the application as a whole, or some parts of it cease to work, however, by using Jenkins, the issues were discovered and resolved in a matter of minutes. The fact that it took us so long to make use of this tool reminds us of the importance of choosing the correct application framework for a project. Ruby on Rails appeared to be the right choice. Specifically, Rails met the security requirements of the project perfectly, had automated model test creation, had a fantastic Gem tooling system to incorporate existing code and allowed for rapid development which suited the project style. However we failed to take into account the platform on which the application required to be run and did not consult tutors or lecturers regarding their knowledge of the application framework. By seeing the evident benefit of the Jenkins tool during the forth iteration we have learned the effect which design decisions can have not only on the application itself but also on the development process. Although we spent a large amount of time deliberating which application framework and environment we should use for our application we did not look at this enough within the wider context of the project, and the consequences which it has outside the customers requirements.

During the fourth iteration we found that the set of tests which had been written during the first iteration and appended to throughout the rest of the project were especially important to the development process. Since the majority of the code being written during this stage was being added above, alongside or even upon existing features, it was critical to know that revisions were not causing issues elsewhere. Within our team we saw a remarkably small number of revisions causing problems to the existing features on the global repository which was largely due to them being discovered by the tests run before committing the revision.

Collaboration on tasks became another strength of our team during this final iteration. Through the ticket system on Trac tasks were still distributed fairly amongst team members, however with each individual having been invested more in certain areas of the overall application throughout the project, it became far more helpful to collaborate on certain new features. This simply involved team members asking for each others help or running through ideas for the specific structure of new code together and was followed up through commenting on Trac tickets to mark the progress of the task. The most prominent benefit that came from such collaborations was the quality of the code written. By working in pairs or sub-groups of team members who each have slightly different perspectives on the implementation, code inserts became more efficient, maintainable and risk averse. This became a more predominant work model during the fourth iteration than it had been in previous. This fact was perhaps due to the nature of the features which were being implemented extensions of models, controllers and pages which had already been written by someone else. However, as the team became more comfortable together and more confident in their own skills within the project there was more space for collaborative work to take place.

When meeting with the customer at the end of the fourth iteration the discussion revealed that as a team we had produced little in the way of helpful documentation for the project. Coming to the end of the project such documentation is essential, not only for end users but to make the application as a whole more maintainable and to map out the development which we as a team have done as well as features which we have not implemented. The fact that this project is a proof of concept makes such documentation even more valuable because it could be used by future teams of developers within the same project domain to help guide their design and the scope of their project. This led us to produce a number of documents including an estimate cost model which describes how we believe the application would operate economically; a road map of functionality including that which we have implemented in our application and some of our design ideas which were outwith the scope of our project; and finally a justification of the design decisions which we made as a team. These documents are an accumulation of minuted discussion both as a team and with the customer along with retrospective thoughts regarding our project.

One issue which we experienced during this iteration was the omission of database migrations in a few revisions committed to our contiguous integration system, Subversion. The database migrations, when run, made changes to the structure of the database. On the occasions when a revision was committed without the migration which was written along side it, the global copy of the application ceased to work due to inconsistencies in what the application was expecting from database and the actual structure of the database itself. Furthermore these mistakes were not flagged up by the contiguous integration tool Jenkins because the application and its tests still ran successfully. This issue was fundamentally a result of team members lacking understanding in the project's use of migrations to maintain a complete correct database. After a discussion together we agreed on the correct use of migrations, and their importance alongside the code.

The final iteration as a whole was the smoothest and most efficient in terms of the software development process, as you would expect. Ultimately it has lead us to realise the importance of the team decisions made on the offset of a large project for example tools we chose to use, each members role within the team and the environment on which the application will be developed. It is then important to invest time in developing these and making them as efficient and effective as possible so that the project can benefit optimally from using them.

4 Conclusions

Developing software tends to be a long and difficult task. However the process can be greatly simplified by the use of a variety of techniques, methodologies and software packages. Through this case study, we hoped to enhance our understanding of the importance of using proper methods, and in the process, we have gained a great deal of insight into working as a team, tackling large-scale projects and dealing effectively with customers.

One of the most important lessons we have learnt from this project is the importance of taking time to plan thoroughly. Taking the time to plan early in the process can help save time in the long run. Even if the plan is not followed precisely, planning is an important part of the process of understanding the user requirements, as well as being necessary for project scheduling. To quote former US president Dwight D. Eisenhower: “I have always found that plans are useless but planning is indispensable.” Extensive initial planning also benefits role distribution for better collaboration and team management. Planning, however, does not stop there but continues throughout the rest of the process. Setting clear milestones and using simple and well defined tickets contributes to an even workload distribution, as well as making the task easier to understand and complete.

During the project we encountered several issues related to our choice of framework for application development. We failed to take into account the development environment we were going to use, which led to us spending a great deal of time fixing issues with the continuous integration tool.

Another useful lesson we have taken from this project is that good documentation is key to successful software development. With many people working on the same project, it is difficult to keep track of what each function does if there is a lack of commenting in the code. Thorough and accurate commit messages on version control systems are essential as it lets the team keep track of the changes that were made in each commit as well as helping to identify the source of any errors when something goes wrong. Further, good documentation is an important form of communication with both the customer and the end-users, as well as making application maintenance easier. It is always a valuable guide for future developers, and this is particularly the case in a proof-of-concept project where the application will be used to aid subsequent designs. Documentation can also be used to outline potential additional features for such designs.

This project also demonstrated to us the benefits of practices such as test-driven development. By writing tests cases before the implementation code, we were better able to track whether the code was working correctly, which parts of the functionality were missing, and identify the origin of any problems.

In addition, branching helped us to avoid merge conflicts, preventing the project as a whole from ceasing to function properly. We were able to test the functionality of a new feature in a separate branch without risking impairing the main project and then merge the changes when we were confident they functioned correctly, fixing any conflicts that occurred. Potential issues can be avoided by committing more frequently, with extensive comments describing the exact feature being implemented or issue fixed.

Central to all the work we did was good communication within the team. This enabled us to pinpoint issues and potential solutions, and ways to avoid such issues in the future, for example by changing how work is distributed by restructuring tickets on Trac, thus improving the team's efficiency. Good communication between team members is important for providing constructive feedback on each others code, e.g. finding errors in the user interface, and for sharing expertise within the team.

To sum up, following modern software development methodologies and practices makes it much easier to gather requirements, avoid and solve issues, and manage time wisely. It is evident from research that many software developers encounter similar problems to the ones we faced. In order to improve the quality of software development across the industry, good practices should be followed and improved upon to move away from old haphazard methods which have been proven to fail.