University of Glasgow | School of Computing Science

Level 3 Project Case Study Dissertation

Team W - Inequalities Benchmarking Tool

Teodor Cholakov
Ivan Kulazhenkov
Alex Pancheva
Daniil Shumko
Ivan Vulkov
David Welsh

7 April 2016

Abstract

This paper presents a case study of the Inequalities Benchmarking Tool, a project developed by a group of six Software Engineering undergraduate students, and reflects on software engineering practices used throughout the development process, relates some of the practices to existing research papers, and suggests possible resolutions.

1 Introduction

This document presents a case study of Team W's Inequalities Benchmarking Tool. It was developed for NHS Dumfries & Galloway and their partners with the aim of providing an efficient and easy to use web-based tool for service providers so that they can determine the number of people from various deprived backgrounds that they might expect to use their service based on its location.

Before any design and development phases started, the team investigated the features and presentation of other similar benchmarking tools. From this research the group encountered a number of features that would be essential to a successful benchmarking tool. More importantly, however, they were able to locate a number of aspects of these tools which were unnecessary and sometimes even detrimental to the customers experience with the software. Most notable of these was the level to which the user is bombarded with various statistical information; namely graphs, charts and tables. There is also a dearth of guidance to differentiate between data, and moreover, a lack of any tutorial for how to navigate and use the benchmarking tool. These downfalls are a serious issue in attracting new users and garnering any kind of positive experience, should they be willing to familiarize themselves with the tool. As a result the team decided to avoid such confusion for the customer and make usability and simplicity the priority for design choices.

The project in itself represents a database of population data from which projected statistics for a multitude of different deprivation indicators can be calculated. Since it was not solely for use by professionals, the tool needed to be simple enough to make data access and extraction clear and trivial for any user. The service provider can select the geographical location of their service, the age group of the population of interest and choose various deprivation indicators for which to generate projected numbers. The results are displayed in an easy to understand and readable table which can be exported as a PDF or CSV file. The administrator of the tool can add new deprivation indicators and add or update the statistics.

The Case Study Background section provides background information about the client and delivers a detailed explanation as to why they need this tool, their goals for the project, and the main achievements of the team.

Following this, the Reflection section is split into iterations and at the end there is a section on general practices that have not changed significantly throughout the process. For each iteration, there are several subsections discussing the main goals for this iteration, the issues the team encountered, how those issues and practices relate to research papers on the same topic, and what should be improved based on what is effective for the team members.

Finally, in the Conclusion section, the team discusses what has been learned during the software process they went through and how those lessons can be generalised and applied to other software engineering projects.

2 Case Study Background

2.1 Customer

There are large inequalities that exist among the whole of Scotland's population. Monitoring an area as large as Scotland is difficult and time-consuming. The customer, the Dumfries & Galloway branch of the NHS, wanted to tackle the inequality issues within the region for which they are responsible. It is vital for the NHS to track the prevalence of deprivation indicators, e.g. poverty, inadequate levels of education, poor access to local amenities, and belonging to a minority group. This is key to the team's objective. To be able to monitor deprived individuals within Dumfries & Galloway is crucial for its health organizations and service providers. To provide the required capabilities, the NHS are approving the development of an inequalities benchmarking tool-kit to make sure that all deprived areas of the community are receiving the support they deserve as well as making sure that relevant deprived sectors of the population are accounted for.

2.2 Preliminary Objectives and Stakeholders

The customer, a representative from the NHS, gave a short introduction to the significance of inequalities across Scotland and, in particular, Dumfries & Galloway. Statistical tools that monitor large urban areas are useful, and can identify large concentrations of deprived people. It is significantly easier to determine areas within towns and cities where the populace is affected by inequality, as well as the circumstances behind it. However, many methods which are useful in urbanised communities are impractical in the countryside and rural areas. These difficulties are further exacerbated by the prevalence of large, sparsely inhabited countryside areas and an irregularly spread out populace. Leaving inequalities unchecked in rural areas can have dire results for the community as a whole; in particular there is sufficient evidence presented by Scottish Neighbourhood Statistics and the National Records of Scotland (NRS) suggesting that the most deprived population in Dumfries & Galloway is almost 70% more at risk of coronary heart disease than the least deprived. The fact that inequalities have such an effect on people's health has led to the NHS and various other large health services becoming key stakeholders in the development of a tool that accounts for inequalities and people from deprived circumstances in Dumfries & Galloway. Additionally, other than the large health organizations it is vital that the benchmarking tool is available publicly so that smaller service providers such as caring agencies can use it for the benefit of their immediate local communities, making them the second and equally important stakeholder.

To cater to the needs of both stakeholders, it is vital that the tool covers the whole area of Dumfries & Galloway, has as large a variety of deprivation indicator statistics included as possible, and is publicly available.

2.3 Aims

The aim was to create an efficient, simple to use, and fully functioning web based tool allowing users to compare their results to estimated numbers produced by the algorithm provided. Essentially, creating a database of expected or projected statistics for a multitude of different deprivation indicators covering all datazones - the standard geographical areas for which statistics are released - within Dumfries & Galloway. The key challenge was to make the tool user friendly and easily accessible, whilst still keeping it detailed enough to provide the necessary functionality for large health organizations. The fact that the tool was not just for professionals meant it needed to be simple enough to make data access and extraction clear and trivial for any user. Notwithstanding the primary objective, the aims of the development can be split into three necessities; covered in more depth within the section below, entitled Initial Planning.

2.3.1 Initial Planning

Information on required deprivation indicators - e.g. ethnicity, council tax bands, and criminal offences - was collected and imported from a variety of different sources, which were mainly web-based. For the data extraction, the team had to investigate the sources provided and analyze the best way to retrieve the data, taking into account different formats for each website and avoiding duplicate data when multiple sites reference the same source or each other.

An algorithm was set up to estimate the number of people from a deprived background from the total population. A range was also included to eliminate any errors due to the estimated nature of expected results. For the back-end, the team had to research the intended framework and try to anticipate any problems that might come up later in the development process.

A front-end user interface was to be created to allow the user to easily interact with the algorithm and database to generate the results that they need. This interface was also required to be styled appropriately, ensuring that it was accessible to all. The development of the front-end had well defined goals and objectives, as the client had a clear vision of the desired "look and feel" and operation of the tool. After the user acceptance testing was conducted, they were eager to help us improve the product and provided the team with useful feedback in order to get the product as close as possible to their envisioned design. The above three objectives were tackled separately and methodically; the project team was broken up into sections, each of which focused on one of the three different requirements: data gathering, algorithm implementation and the back and front-ends.

2.4 Requirement Changes

Throughout the entirety of the development phase regular meetings were scheduled for further requirements gathering to make sure that the benchmarking tool was taking shape in the right way. Most notable of the developments following the customer meetings were aspects of the design of the front end that should be improved in some way or changed. The functionalities considered useful for addition during the development cycle focus on 3 distinct areas of the tool:

Data Visualization - There was constant debate and improvement regarding data and output visualization. The tool gave the user the ability to narrow down their data search as little or as much as possible. Above all, the visual representation for the size of geographical areas and the format of the final output were changed significantly for the user's benefit. The geographical area selection form was changed to make it possible for a user to select a variety of regional sizes from the smallest datazone all the way to Dumfries & Galloway as a whole. The format for the tool's output was modified to include a bar chart stating the ranges and the expected number for a each selected deprivation indicator. The thought process behind this was to provide the user with an option for quickly consulting the graph rather than studying the output from the data table.

Result Download - Another product feature under constant scrutiny was the way in which the tool's output was to be exported. After various discussions and changes it was decided to keep the output primarily within the tool. However, an option was added for the user to be able to export and download the data in both csv and pdf formats. This gave the tool a flexibility in terms of being portable as the data did not rely on a connection to the server.

Update of Statistics within the Database - A very useful optional functionality under regular consideration was how updatable the tool was, making an allowance for the irregularity and possible future inconsistency of data from the sources. By including an administrative page for the website, the ability to manually add both individual deprivation indicators and the ability to upload data in CSV files was incorporated into the final product.

2.5 Final Product Features

Following the requirement changes, by the end of development a final working tool was produced as outlined by the customer during the planning phase. The outcome was a lightweight application performing all expected tasks as required by health service providers.

The benchmarking tool's main function, estimating the number of deprived people from a specified sample size, is just the bare skeleton of the its functionality. A rich form is provided to allow users to enter the details about their group, including the typical age range and sex and the geographical location. The location selector allows users to choose from various granularities of areas. Users can then select as many or

as few deprivation indicators for which they want to get results. Additionally, the tool gives a concise yet meaningful introduction as to its functionality and guidance for use. A detailed guide has also been included explaining the creation of the algorithm used, as well as the optimal way to interpret its output. The applications simplicity and supervision provided by these tutorials mean that any user - no matter what level of knowledge they have - can successfully use this tool.

The flexibility of the tool complements the diversity of the health industry and the huge variety of organizations the inequality benchmarker will be serving. The tools outputs are portable and not reliant on a web connection due to the ability to export results in various formats. The time stamps included with the downloaded documents ensure that data is not redundant and valid records can be maintained. Along with the numerical representation of data, a graphical visualization of the result is also included; color coding points of interest, namely the ranges and expected number of deprived persons.

The final critical feature included was the administrative panel. Anyone with administrative rights is able to add new deprivation indicators along with their assigned categories. Moreover, this panel allows the update of data inside the database. Upon new data for the Dumfries & Galloway region becoming available, it is possible to upload the new files, making updating as hassle free as possible. However, given the irregularity of new data being published the inconvenience of this is rather minor.

3 Reflection

3.1 Iteration 1

3.1.1 Goals

The main goal for the first iteration was for the team to familiarise themselves with the project description and to focus on the two main elements, specifically understanding what data needed to be extracted from the websites provided and research the algorithm in order to implement it. At this stage, the team was already familiar with some aspects of the Trac ticket management system being used, but, as can be seen from the tickets created at that time, the descriptions of the tasks were rather vague and the lack of understanding of what the team was meant to do led to setting too many goals for the Second Customer Meeting. Initially, the plan was to have data extracted from some of the websites. Later on, it became evident that data extraction without a good understanding of the data is not a reasonable approach. However, this was not something the team thought about when discussing the plan for the second iteration with the Customer, since at this stage the list of websites wasnt available. Furthermore, during the First Customer Meeting the team got only a general understanding of the project requirements.

3.1.2 User Stories

Documenting requirements facilitates work prioritisation and planning. One of the main activities that the team undertook during the first iteration was creating user stories. This helped to give a better understanding of the problem domain and the features that needed to be implemented. Although the user stories were high-level goals and each of them had various components the team needed to focus on, this activity proved to be beneficial and outlined the required features. The team followed the INVEST strategy for writing user stories as suggested and this resulted in well-documented required features and led to further discussion of some of the features at the next Customer meeting.

3.1.3 Technologies Used

Spring Boot:

In this iteration, discussion over which framework to use for developing the project was held, with the main options being Django[**?**] and Spring Boot[**?**]. On one hand, everybody had used Django, so no time would be spent learning new technologies. Django is easy to setup and lightweight and provides the needed modularity, a lot of useful features such as a built in CMS Admin Panel, and an integrated support for templates.

However, although Spring Boot is a framework the team was unfamiliar with, it is Java-based and has extensive documentation, so it was assumed that learning it would be trivial. It is also quite lightweight, even compared to Django. There is also minimal boilerplate code, which makes the application small and easily understandable. Another major advantage is the more efficient and configurable back-end. Considering all of these, the team decided it would be best to use Spring Boot, as it would not be much harder to setup and build the tool, but it would be much easier to maintain, and is better suited for open sourcing, should the team decide to so.

SQLite:

As for the database system, the team decided on using SQLite as it is lightweight, easy to use and easily portable. It works well for low to medium traffic websites, such as this tool, as it can handle more than 100,000 hits per day with ease. In addition, every team member had used it before.

At this stage, the front-end design was not a priority. A simple template was used to provide the basic functionality.

3.1.4 Team Organisation: Sub-teams

As the initial goals were quite vague and the team members didnt know each other very well, the team communication during the first few weeks was quite problematic.

For example, the tasks were not equally spread or sometimes two people were looking at the same websites. There were times when some team members did not know what to do. Given the problem description, this issue was partially resolved with the decision to split the team into two sub-teams; one to work on the database and another to work on the algorithm.

The main goal of the database team was to research the data that needed to be extracted and design a suitable database to store it. A good understanding of how the geographical hierarchy to be represented was required but since the team wasnt aware of the geographical details at this stage, the initial database design was rather nave. The design of the database changed throughout the process as better understanding was gained. Since some of the sources and deprivation indicators were known at this point, the database team developed tools to extract and reformat the required data. One major setback the team encountered at this point was duplicate data and different data formats. Some of the sources were referencing each other (most websites were using data from the Census) therefore just downloading and reformatting all the data was not feasible. Additionally, data formatting was website-specific, so creating a universal data extraction tool was not possible. Instead, the team opted to create a simple read and format CSV tool and tweak it for each website.

The other main focus was the algorithm. The proof of concept that was provided by the customer was of significant help, so creating a simple program with command line input for the algorithm was straight-forward. As the team had decided on what framework to use already, Java was used for the implementation.

3.2 Iteration 2

3.2.1 Goals

After the team had acquired a good understanding of the project and asked various clarification questions, the main focus of this iteration was to start data extraction and populate the database with some of the data. Some of the work during this stage included collaboration with other teams. The decision to use SQLite during iteration one proved to be premature and the database had to be migrated to H2.

3.2.2 Collaborative Data Extraction with Team R

One of the challenges for the second iteration was getting the data for the number of households in different council tax bands from the Scottish Assessors Association (SAA). Their website does not provide any APIs to get the data programmatically, so the team decided to work together with Team R to develop a crawler that collects data for all the council tax bands in Dumfries & Galloway. The crawler works by iterating over an array with all the postcodes in the region and sends a request to the SAA site for the council tax band of each household within each postcode. After that,

it crawls the page to find the number of households in each council tax band. The result was written to a CSV file in order to easily import the data into the database.

Working with another team came with some unexpected issues. Before the work on the data extraction script began, the two teams did not negotiate any terms of ownership. Neither did they discuss who would take the credit for the final product of their combined effort. This could have been a problem because the client requested a copy of the results. In the end, both teams agreed to send a copy each and this is how the situation was dealt with. However, the team realised that for future collaborative work with external members, there should be a clear understanding from both parties of ownership and responsibilities. Discussing the terms and conditions before any work begins will clear any possible misunderstandings allowing both parties could focus on actually solving the problem at hand.

3.2.3 Database Migration

When it came to integrating the different parts of the project, it turned out that the team's decision to use SQLite for the database system was premature. The Hibernate framework that controls data in Spring Boot does not work natively with SQLite. While it is possible to customise the framework to allow this, the team felt that this would not be a worthwhile exercise as there was nothing in the database structure that tied it to SQLite and using a native Java based database system would allow a more standard system to be built. Once it had been decided that a new database system would be used, the team started to research the alternatives. The standard systems used with Spring include H2, Derby and HSQLDB, all with native support in the Hibernate framework. Eventually, H2[**?**] was chosen as it is typically the system recommended for development in Spring Boot and its features closely matched those of SQLite.

The migration of the database from SQLite to H2 proved to be fairly trivial. Both SQLite and H2 use a fairly standard SQL grammar with only a few of the database creation statements needing to be altered. The structure of the database remained the same through the migration.

In this case the change to another technology was trivial and this premature decision did not really have dire consequences, so the overall process wasnt affected. However, a delayed commitment approach could have been utilised to avoid these issues. Delaying commitment, as described by Harold Thimbleby, involves avoiding making firm decisions about a software system until they can be made with an informed approach, allowing for the constraints of one choice to be discovered before a commitment has been made [**?**].

3.2.4 Technologies Used

Apart from the database migration, another significant technological decision the team made during this iteration was choosing Thymeleaf over JavaServer Pages

(JSP). Initially, the team decided to try JSP but as the team came across Thymeleaf[**?**], its detailed documentation, better structure, and easier back-end data validation led to the decision to use it over JSP. Both JSP and Thymeleaf were new to all members of the team, so it was important to go for something that could be easily learned in a short period of time. This was an important factor because the focus of this iteration was to create a working version of the software in order to be able to perform acceptance testing and there were many features to be developed over the period of two weeks.

3.3 Iteration 3

3.3.1 Goals

The main goal of the third iteration was to deliver a fully working product suitable for testing. In order to achieve this, the user was to be able to test the main feature of the application: inputting the required data and getting the estimated number of people and the ranges.

3.3.2 Inter-team Communication and Merging Option

During this iteration, the customer suggested all teams to work together on the application since more features could be implemented and it seemed like efforts had been duplicated: all teams were trying to extract the same data and get the algorithm working. This time could have been used to develop other features such as a more easily understandable geography selection.

This decision was quite vital to the work of all teams and also had a serious influence on the product delivered to the customer at the end. An informed decision was required, pros and cons of the two options had to be discussed in detail with representatives from the four teams. This decision was then scheduled to be communicated with the Customer at the video conference that took place between the Fourth and the Fifth Customer Meetings.

At the meeting, it became clear that the four teams were at different stages of their development process: some had most of the data extracted whereas others were working only with a few deprivation indicators but were focusing on the front-end and the user experience. Each team had preferred different technologies: one was using Spring Boot with AngularJS for front-end and other teams were using Django. Since different technologies were being used and each team was having internal problems related to task distribution, having 24 people working on the same project, even if new features were being added, would have caused problems since the customer wasnt planning to make substantial changes to the project requirements. Furthermore, there was only a month left before the Final Demonstration and certainly that was a major reason to decide against the possible merge. Unfortunately, the four teams ended up

not working together at all, excluding the collaboration between Team W and Team R at the initial stages.

This situation should have been approached differently. The merge option should have been discussed earlier and would have been viable if it had been discussed, for example, in December, before everyone had a working product. This would at least have ensured that there was minimal duplication of efforts and more attention would have been paid to certain features that are not as well implemented as they could have been. Furthermore, inter-team communication would have provided a better understanding of the project at the initial stages of requirements gathering.

3.3.3 Pair Programming

During this iteration, some of the team members decided to try pair programming. Sommerville [**?**] notes that this is an innovative and controversial practice because it may lead to both better code quality in some cases - for example, when inexperienced programmers are working on a project - and loss of productivity when the pair consists of more experienced developers.

However, the team found it to be highly productive and this practice certainly improved the code quality. Pair programming was tried when designing the script for some of the data extraction and when team members were trying to link the Spring Boot backend, the front-end, and the algorithm. In both cases, there was a discussion about how it should be done which prompted the pair to come up with the best way to solve the problem. The effect pair programming has on learning experience and problem solving has been discussed in a paper by Alistair Cockburn and Laurie Williams. William's experiment with her university class in University of Utah confirms that pair programming tends to be beneficial for students and it enhances learning. 84% of William's class agreed that working with a partner helped them learn the required topic "better and faster"[**?**].

The team also found that pair programming was a useful tool for motivating participation in the project and task completion. When two people were working on the same piece of code, the task was completed prior to the deadline and it seems that each of the team members was affected by peer pressure. Sommerville [**?**] also noted that pair programming supports the idea of collective ownership and responsibility for the system and this also relates to Weinberg's idea of egoless programming. The code did not seem to be the responsibility of only one person and both people in the pair were willing to make any additional changes to the code.

Furthermore, Williams and Cockburn explain the importance of pair programing for the overall communication and team building: "If the pair can work together, then they learn ways to communicate more easily and they communicate more often. This raises the communication bandwidth and frequency within the project, increasing overall information flow within the team"[**?**]. In fact, this is something Team W has noticed as well. Those team members that worked in pairs were most likely to talk about the project outside the regular team meetings and the allocated Facebook

group. This additional communication proved to be beneficial when members were trying to understand some of the requirements and in preparation for the customer meetings.

Despite the fact that pair programming worked really well for some team members, not everyone went through this process, so the overall result was affected. However, the code reviews that had been conducted later on helped with refactoring the code and fixing software errors. In terms of team communication and given the team structure, having a full cycle of pair programming would have improved the tasks distribution and the flow of information between team members.

3.3.4 Project Management Tools

The Trac ticket management system seemed easy to use and the main features were utilised in the beginning. However, it took a few months until the team began using it effectively.

An example that would have definitely improved the communication in the initial stages is having more detailed tickets that the team members accept once they are working on the particular feature. The workflow during the first two iterations wasnt well-defined and often what some team members were doing was to open the ticket and directly close it when the task is completed. This poor practice resulted into two team members working on the same task simultaneously and duplicating efforts without being aware of what the other person was working on. An easy way to solve this problem would have been to agree on how to use Trac more effectively and have a guide on what is encouraged and what should be avoided. However, by the time of the third iteration, everyone had a better understanding of how to use Trac efficiently: tickets were more specific and the team members were using the ticket ownership options correctly.

Another aspect of the use of project management tools that improved was the cross-linking of tickets and Subversion commits. This is certainly a practice that makes it easier to keep track of progress and encourages meaningful commit messages.

3.3.5 User Acceptance Testing

In February, between the Third and the Fourth customer meetings, the client organised a User Acceptance Testing event with approximately 20 people from various backgrounds (representatives from the NHS, local councils, care homes, the Crichton Institute, housing associations, etc.) present. This event can be considered as an example of Beta testing, since as Ian Sommerville [**?**] describes the process, the unfinished released was made available to a larger group of people, the potential early adopters of the Benchmarking Tool. This event brought to the attention of the customer improvements that they hadnt considered before and clarified certain aspects.

The feedback that was provided after conducting the User Acceptance Testing (UAT) was mostly positive. The testers found that they intuitively knew how to use the website, which was the one of the main goals of the project. A number of improvements were proposed as well. At the time, the user would select a datazone from a long drop down menu with the names of all the datazones. The testers suggested implementing a more understandable datazone selection, so that it would be easier to navigate and to provide more meaningful options. Another suggestion was to allow for users to see more than one indicator in the output report at a time as well as being able to export the result to either PDF or a file readable by Excel or similar programs with a timestamp and sources provided. Although the tool was easy to use, the client proposed that a user guide be added for the end-users as well as pages with detailed explanation about how exactly the tool works and how it generates the results.

Overall, the style and look of the prototype was liked by testers and was close to what the client had originally intended when the project started.

3.4 Iteration 4

3.4.1 Goals

The main focus of this iteration included finalising the product and working on the administrative features. An understandable geography selection was also to be added as well as visualisation and extraction of the results.

3.4.2 Scrum

The main idea of Scrum meetings is to make the best use of the time and resources available. While the duration of the sprints has been determined independently and the end-of-iteration retrospectives were a requirement, the team didnt really manage to properly implement the organisation of the Scrum meetings (which would have been weekly for this project instead of daily). As Jeff Sutherland and Ken Schwaber - the co-creators of the Scrum framework - explain in their paper, one of the main goals of the framework is to make sure theres transparency and everything happening during the project is visible[**?**].

The consequence of these weekly Scrum meetings is evident improvement in the communication between team members. However, as it can be noted from the conducted retrospectives, the team communication had its ups and downs and while there was a significant improvement during the last iteration, the whole process could have been made more effective from the early stages. During the last iteration, the team members became better at having a short meeting in the beginning before everyone starts working on something and stating what has been done and what they are planning to do by the next team meeting. There were a few attempts to conduct Scrum meetings during the initial stages of the project but due to members being late for meetings

and tasks being assigned mostly via Trac, there were some weeks without those short Scrum meetings that aim to enhance visibility.

The role of a ScrumMaster is there for a reason since there should be someone to guide the rest of the team and to ensure that the Scrum framework is being followed. One possible improvement in future projects would be choosing a ScrumMaster whose main responsibility would be making sure the team is using this agile framework effectively to achieve better level of visibility and team communication.

3.4.3 Team Organisation

One of the external software process requirements was to assign roles to each team member in the beginning of the project and those roles should have been decided after a discussion of everyones strengths and weaknesses. Following the initial specifications, the main roles within the team were split as follows: project manager, customer liaison, chief architect, librarian/secretary, testmanager, and tool smith. While this worked well for some aspects of the project, such as customer interaction and design decisions, this wasnt the case for some of the other team roles.

During some of the iterations, the team ended up following some of the aspects of the Weinberg's decentralised democratic group that promotes egoless programming with group leadership being a rotating function, becoming the responsibility of the individual with the appropriate skills. As Marilyn Mantei explains in The Effect of Programming Team Structures on Programming Tasks, the Weinberg's Decentralised Structure has several weaknesses [**?**] as the team found out. The first one is related to deadline driven task completion: some team members ended up postponing the complete implementation of some features for future iterations. One of the examples is the team's intention to have the UI implemented by the second iteration but what happened in practice was that it was done for the third one. While the decentralised team structure encourages communication between members, as there is twice the communication compared to a centralised team structure, and for some of the team members this has been beneficial (See Pair Programming), it affects task completion and task distribution. There were open tickets without owners that were only completed as the deadline approached by a team member who noticed that they were not going to be completed on time. Furthermore, the team wasn't really able to implement the rotating leadership function due to time-constraints and the equal level of experience. The aspects where experience was vital were taken under consideration: the choice of customer liaison and chief architect.

Another option would have been Baker's chief architect model in which decisions are made at the top level. The success of the team in this case depends on the leadership and communication abilities of the chief architect and as the paper points out this structure works well when there is an experienced chief architect [**?**]. This is related to the option of controlled team organisation whereby a team leader distributes the available tasks. However, in the initial stages, the team lacked a member who was willing to stand up and take over the responsibility of being a team leader. The lack of team organisation became more obvious towards the end of the project when final

touches were needed and some members didnt have enough to do while others were overwhelmed with work.

Based on what the team has experienced, it seems like a controlled decentralised system or a variant on this would have been a better fit. The task distribution would have improved with the presence of a team leader. Furthermore, since the sub-team organisation worked well in the initial stages, this could have been done later on as well by having one team to work on the front-end functionality (e.g. extraction of results, geography selection, visualisation of results) and another on the back-end functionality (e.g. database, data extraction). It would have been much easier to split tasks within those teams. Pair-programming could have been done within those sub-teams as well.

3.4.4 Code Reviews/Software Inspections

Performing code reviews is one of the main practices used to ensure that high-quality code is produced. As Ian Sommerville points out [**?**], the code reviews might be both an individual (the author should write well-organised code and make sure the code works before committing and is responsible for organising code reviews) and a group exercise. However, it was only during the fourth iteration that the team started to organise code reviews. According to Fagan, software inspections are an effective way for defect detection, and during the software inspection that the team performed, several refactoring options were found. One way to improve this aspect of the team's work during future projects would be to organise code reviews frequently, especially when core features are scheduled for implementation. Furthermore, Fagan describes the structure of the inspection team which benefits from the role of an inspection moderator [**?**]. Similarly to the problem that has been encountered with the lack of a ScrumMaster, the fact that no one was in charge of planning and organising inspections resulted in the team scheduling only one. However, even though the above mentioned inspection wasnt organised in the recommended way, in addition to discussing the design and the implementation, identifying code that should be refactored, and updating the documentation, the team used the Metrics Reloaded Plugin for IntelliJ to analyse aspects such as coupling between objects and lack of cohesion of methods.

3.4.5 Results Visualisation and Download

One of the last features that the team implemented was the results' export functionality and the visualisation via a chart. Various JavaScript libraries were used in order to achieve the task. The end result provides the user with the ability to export the result of their query as CSV or PDF files. Moreover, they are able to see a detailed stacked bar chart if they would like to.

The main challenge here was understanding how the customer wanted the results to be visualised. During the first requirement gathering session, this aspect of the

project was not discussed in depth, so during the last sessions with the customer the team had to put more focus on this problem. However, the development of the download feature started before these last sessions. The motive behind this was that once the ability to extract some data from the results page of the benchmarking tool was implemented, it would be easy to adjust how it is presented according to the customers needs. At the time it seemed like a reasonable decision as all other high priority features were already being worked on by other team members. The team had the spare resources and allocating them on this feature early on could have saved some time towards the end of the project. However, making the wrong assumptions about some of the customer's requirements led to developing a feature (extraction of results to PDF) that had to be changed entirely near the end of the project. Originally, the data in the PDF was presented as a dictionary key-value pair. After meeting with the customer, the team was told that a table would be a better representation of the results and the feature had to be modified. Creating a table in a PDF file using JavaScript turned out to be harder than expected. The team tried several different approaches and in the end the jsPDF-AutoTable library was used [**?**]. Had these requirements been understood earlier, more time would have been spent to find the best way to implement that feature. Moreover, all the time spent on creating the key-value pair PDF representation was effectively wasted.

Another feature that was discussed towards the end, and was modified several times, was the data visualisation. Initially, the customer described it as a feature the tool could have if there was time to implement it and the team hadnt considered this feature until the last customer meeting. During this meeting, the customer suggested having a spine chart next to the results table. However, after careful consideration of the effect this type of positioning and visualisation had on the end user, the team member who was developing this feature came up with the idea of a stacked horizontal bar chart that was displayed underneath the results table. Considering the misunderstanding that occurred with the extraction feature, the implementation of the visualisation was confirmed with the customer and they suggested including a dynamic title and renaming some of the fields.

Understanding the requirements before the beginning of the development process is crucial. The team had the best intentions, but those wrong assumption turned out to put hours of work to waste when developing the PDF extraction. On the other hand, sometimes going against what has been suggested by the customer could give better results as it may offer a solution they hadnt considered. In this case, the changes should be discussed in order to make sure that the best possible solution is applied and the customer requirements are met.

3.4.6 Integration Testing

At this stage, integration testing was conducted to determine whether all the features work as intended.

Since a major part of the project was database access, testing the Database Access Layer was the main focus. All the Database Access Objects (DAO) were tested

in full using an embedded database to perform mock queries. Some minor bugs were discovered and patched. Another issue that was discovered was that the input form for the population was not sanitized, thus making the tool vulnerable to SQL injections. The solution was to parameterize the form and validate the input, so it does not take any string as a valid query.

Testing and validation of REST services in Java is harder than in dynamic languages such as Ruby. This is why the team used the REST-Assured Framework to do so [**?**] It provides an easy to use and understand API that brings the simplicity of dynamic languages into the Java domain. Combined with the Hamcrest library [**?**] of matchers, the creation of flexible expressions of intent in tests was made considerably easier.

3.5 General Practices

3.5.1 Role of the Customer Liaison

One of the main roles within the team was that of the customer liaison. The person responsible for this aspect of the project was identified in the early stages, before the First Customer Meeting, based on past experience and interests. As Prof Elke Hochmller discusses in The Requirements Engineer as a Liaison Officer in Agile Software Development, "continuous and close communication and collaboration" with the customer is vital [**?**]. However, sometimes the customer is not able to identify all the requirements or identify them correctly. Furthermore, it is easier to channel the communication. Having someone with previous experience in this area helped shape the communication with the customer. While the whole team attended the initial requirements gathering meeting which helped to identify the main features of the application, the customer liaison was in charge of communicating with the customer when further clarifications were required or to discuss the implementation of certain features, e.g. NHS style guidelines, the design of the chart, PDF style clarifications and information for method and about page. As the paper explains, the customer liaison role aims to facilitate the work of the developers and the customer and from what the team has noticed, this had a positive effect on organisation and the effectiveness of teamwork.

3.5.2 Customer Interaction

The customer interaction throughout the course of the project can be described as effective, excluding a few minor issues. The first is related to the list of deprivation indicators that the team was given. During the first few iterations, the team was given a list of deprivation indicators and sources that was quite vague and there were problems with the name of the deprivation indicator and where to get the information from (the names of some deprivation indicators werent matching the Census files, some data was available from multiple resources, etc.). These problems required multiple clarification emails and getting the data extracted took longer than expected. However, after the video-conference, the customer provided the team with

a detailed list of deprivation indicators with specific resources that facilitated the process. This is something that should have been requested earlier and it would have helped with the initial stage of data extraction and reduced the efforts.

Another aspect, related to the customer interaction, was the suggestion to merge the four project into one. The reasons why this was not possible and the justification of the decision were discussed earlier. Its important to note here that as soon as the four teams explained to the Customer why they were again against the merge, they demonstrated understanding and continued to support each team equally.

3.5.3 Retrospectives

One of the practices that has been encouraged was conducting retrospectives after each iteration. There was a recommended way that included identifying what the team liked, learned, longed for, and lacked. After identifying those areas, team members were meant to propose solutions and implement them during the upcoming iteration. The process of conducting retrospectives improved with time, in terms of what team members were adding to the different categories. Everyone learned that retrospectives should focus on software engineering practices and how they work as a team rather than on the technical aspects of the project. Certainly, conducting retrospectives improved the software process: tickets started reflecting smaller and more detailed tasks (there was a substantial increase in the number of tickets from 8 tickets for iteration one to 58 tickets for the last iteration). Team members were constantly trying to resolve their communication problems and the improved use of Trac helped to an extent.

Ideally, before discussing the aspects that went well and identifying areas for improvement, the team members should take time and independently think about their experience during the corresponding retrospective. Contrary to what was described, the team members started their retrospective with a discussion. The reason this practice is not recommended is the effect other members have on each other's ideas. Furthermore, some people may not feel comfortable saying what the problem is if it is related to another team member. Certainly, the procedure for conducting retrospectives should be revisited in future projects.

4 Conclusion

The Inequalities Benchmarking Tool was the first large scale project undertaken by the team and as this was the team's first agile software project, the team members not only needed to learn how to use certain technologies they hadnt encountered before - such as Trac, Subversion and Jenkins - but also needed to learn how to put agile practices into effect efficiently and make the best of them. However, as it could be expected, the software process team W went through wasnt ideal and there were many aspects that could have been improved. The main problems and lessons can

be split into a few categories: design and technology decisions, team dynamics, and techniques for process improvement.

One of the main issues, as discussed in Database Migration, was the database migration from SQLite to H2. This problem the team faced was caused by the lack of research and in-depth understanding of Spring Boot and the database engine support it has. This could have been approached differently and although there werent any serious consequences for the team, in practice, when design and technology decisions are made, there should be a discussion of motivation and justification for those. In this particular case the change was trivial but sometimes it could take longer to migrate from one platform to another. When a team is trying to decide on technologies, the different options should be carefully researched and compared because if the team doesnt spend time on these initial stages and doesnt make an informed decision, this could affect the later stages of the project.

Team dynamics, namely team organisation and communication, affected the work of team W. The team managed to try different team organisation structures (see 3.4.3) but due to the limited time, the most efficient one, controlled decentralised structure, wasn't implemented. The team organisation affected the communication and the task distribution for the entire project. Although the team ended up completing the task successfully by creating a fully working application with all required features required by NHS Dumfries & Galloway, the work was not equally split and at times when some team members were feeling overwhelmed, others did not have enough to do. However, pair programming proved to be not only an effective tool when it comes to code quality but it helped facilitate the communication between team members.

Despite the evidence of process improvement, there are aspects that should have been discussed in-depth during the retrospectives and better solutions should have been suggested. The issue of team organisation should have been raised earlier as well as task distribution. Furthermore, it is vital for team members to have a good understanding of how to conduct retrospectives from the beginning because it can be argued that some of the practices of the team - e.g. starting a retrospective with a discussion rather than giving time to team members to reflect on their experience on their own - affected the results and the effectiveness of the retrospectives. Overall, considering the successful implementation of the Inequalities Benchmarking Tool and the indisputable improvement of the software process from what it was in the first iteration, the team is satisfied with what has been achieved. Furthermore, they are aware of what did not work well and now after completing the project, the team members can reflect on their project experience and learn from it.