

Use of Metrics and their Visualisations in Software Engineering Projects

Edward Armstrong*

Neville Churcher

University of Canterbury Computer Science Department

*epa31@uclive.ac.nz

Abstract: In this paper, we describe an observational investigation that explores whether creating visualisation systems using software metrics promotes reflection. We presented metric visualisations to six groups, in separate sessions, from an undergraduate software engineering project course, each consisting of 7-8 students. The results and feedback from these sessions were used to build upon the visualisation system to improve them. Feedback and recorded statements from the students showed that the students reflected but were primarily focused on performance in their course. Delayed questions to the groups showed that the engagement was increased in all groups. It was found that for the visualisations to be of use to the students they must provide direct information about the student's code base, such as telling students where to look to find possible faults. This information is then used to create a visualisation system which looks to address the issues students had with the initial implementation.

Introduction

Aims and Objectives

This paper looks at the creating a system to guide students in a software engineering project course while they are designing and implementing their project. In the first section we work with students while they are building a yearlong group project. The input we use for the system would be each group's codebase.

As there are no set rules for creating "good" software but instead fuzzy heuristics, the modelling of such a domain is very difficult.

The first step in achieving this goal was to gather frequently used metrics and building simple histograms based upon these. Using these, an observational investigation was conducted with the students, where the histograms were discussed and the comments of the students recorded.

After receiving feedback from the students we create a system of software visualisations that looks to deal with the issues students had with the previous system. This involves a set of three interactive web visualisations.

Relevant Background

OSM and OSSM

Because we are working with students and looking to engage them in social learning using the visualisations, we look at using open student modelling (OSM, also called Open Learner Model or OLM) and the closely related Open social student models.

Open student model is used to help students to learn by encouraging them to reflect on their own work and knowledge through providing extra information to the student. In this paper we look at an open student model as being a computerised system that aids the learning of a student. The open student model is a set of elements that characterise a student's progress, or any other such statistical change, in comparison to themselves. This model has the potential to record detailed statistics about the student and their particular knowledge of a specified domain, however, we view this as an optional aspect of the model. When a system or part of a system makes decisions about a student's learning path to make it specific to the given student, it is using the student

model. An example of this is basing the user interface and what is being shown to a user based upon the knowledge/skill level (the state of the student model) of that user.

When a computer makes such decisions it uses the student model, and as such the model may naturally be in a computer readable state and not in a human readable state. If this is the case, a student will not be able to view the model. Allowing the student to see the model, requires us to present the model to the student in a format that allows the student to easily process and understand it. Though opening the student model to the student we help metacognitive processes such as self-awareness [1].

Open student models have been previously researched, and shown to be able to improve learning, though the demonstration of progress that a student has had in a given domain [2-4]. The important aspects of this are the comparison of a student's belief on their own knowledge with comparison to the belief a computer system has about the students' knowledge and how the student reflects on their own knowledge with relation to themselves. If the student is able to understand the differences between the computer systems model and what they believe their own model to be, it allows the student to reassess their knowledge. This could be in the form of better understanding the particular domain, building a better path through their learning of the domain and possibly making new connections between concepts relating to this domain.

We know that a student model by the system will not always be correct, and there are a few times that this will probably always be true. We can see this when a model is first initiated, where the student model will have to be empty or some average of the population used, as we have no knowledge of the particular student yet. The presentation of the model to the student from the beginning allows a student to view and assess the model that the system has of the student. This allows the student to see how an action to their model based upon a change in knowledge will change the systems' beliefs.

In this paper we do assume that the student model is valid and that other research has looked into confirming the validity of such a model. The effectiveness of an open student model can be broken down into two sections: what is being displayed to the student and how we display this information to the student. We look at both in this paper. For a task which is well defined in comparison to a task which is not well defined we should usually find that it is easier to define the information being displayed (the "what"). Finding well defined tasks is easier in well-defined

domains [5]. For instance, if we look at the domain of running which is a subdomain of physical activity, a well-defined domain, we can use concepts such as speed and distance to evaluate upon. We would be able to track a student according to these concepts and rank their progress in a simple manner. However, as the tasks in a domain increase in number and in complexity they become harder to define and harder to define what we wish to display to represent these tasks. Therefore, as the skills needed to complete a task and the number of classes increase we find it harder to measure the domain. We categorise software engineering, as a set of hard to define and therefore measure, tasks.

The displaying of the open student model to the student is also of interest to us, as we will be improving this design with relation to the tasks in a software engineering group project. Representation of open student models has been widely investigated and in many forms. Kay [6] describes a multi-tier modelling representation and toolkit which gives users an understanding of their own user models. The user model there uses different forms to represent concepts. In contrast to the difficulty of measuring the software engineering tasks, visualising techniques merely represent the data collected in the hard step in a visual way with minimum data processing.

Tools for the visualisation of user models are available [7], which explores visualisation methods to present the data, such as colour, shape and size, which we also explore in our visualisations.

An open social student model (OSSM) makes use of an open student model, which is either individual or aggregated, which is then used and shown to other students in the same course [8]. Allowing a student to see the OSSM, means a student is able to make a comparison against other students, based upon the statistic the OSSM is measuring, and rank themselves accordingly. The Achievement-Goal Orientation looks at two approaches of students in a learning situation, these are, a performance oriented approach and a mastery oriented approach [9]. The mastery approach is defined as a motivation towards mastering a given skill or concept for the focuses such as solo learning and the performance approach is driven through a competitive drive, focusing on their scores in comparison to other students. A student will likely show exhibit features from both approaches, however, one may be favoured. There has been work toward increasing the benefit of both approaches. OSSM is categorised as a performance model, this is due to the comparison

of one model against another based upon a given statistic. OSSM can be used in keeping performance oriented students engaged for longer [8].

The Domain and Modelling It

From the past section and general knowledge, we know that software engineering and specifically the development of software as a task is extremely complex, involving using multiple skills to complete multiple tasks. Software development as used here is the total process involved in engineering a piece of large scale software, not just the process of coding or designing the software.

During their time at university, students in software engineering and computer science, will learn a number of concepts related to computer science and here we look at the concept of metrics. Metrics are a function, that when run over a code base will output a measurement which in some way abstracts the view of the codebase. They will should also be an objective and repeatable measure of an aspect of the codebase. In the students' case, metrics are viewed as tools to help them understand other concepts in the field. We can view how this works by using an example with the concept of software testing. In software development testing is used to guarantee a section of software works as intended. And if we look specifically at one testing method, Test Driven Development [10], we are able to see that we require the skills of using unit testing to drive an implementation of the software. We can do some measurement of the testing to gather information about the amount of code that is “covered” by the testing performed. This is a metric called “test coverage” and is a simple metric which can be easily learned. However, if we are to try to use this metric to convey how well tested our software project is we may impart false confidence in our product. This could be as the tests created do not test to ensure correctness of the project and rather test to increase the amount of test coverage our project has instead. For testing to be of use in a software project we must ensure that actual functionality is covered to the highest degree possible, reaching from the core cases to as many edge cases as possible. If we go for a high test coverage we do not necessarily meet these demands. We cannot actually measure test quality overall, instead the developer must instead use good practice and their own judgement in creating the tests.

Because of the want for descriptive metrics that provide an accurate measure for the many aspects of software development, a large number of metrics have been proposed. For example, Lines of Code (LOC) is one of the simplest metrics about a code base, simply returning the number of lines of code written and as above Test Coverage is another metric. Using another example, a developer may wish to see how complex their codebase is. Complexity is important as it can affect other aspects such as maintainability and reusability of code. So the developer may use metrics such as lines of code, NPath or Cyclomatic complexity [11]. The problem we face here is, due to the complexity of software we must look at each case of software individually with respect to metrics, meaning each can only be evaluated on in the context of the software that is being examined. We do not know if there is a specific value for complexity that any given piece of software cannot exceed, rather we must use contextual judgement to create our own thresholds. Other aspects of software development fall on a scale of complexity and can be harder again to measure. Therefore, all metrics require significant evaluation in context to achieve interpretable results from which actions can be taken. We know that there is not a single method for software development but rather many incorrect ways. And in general, the continual observation of all aspects of the development process should be undertaken to achieve good quality software. Combining all the above points makes for teaching, learning and modelling a software engineering project course difficult.

Visualisations for Modelling Software

The subject of visualisation of software code bases has been explored extensively. Recently there have been a number of papers which explore the use of visualisations to support code smell inspection [12-14] and these have proven that the visualisations improved the user's ability to find defects.

There are also a number of static analysis tools that can be used to find bugs such as [15-17] which look at examining the code without running it.

Our implementation does not push the bounds on sophisticated visualisation techniques and rather the techniques used are guided, specifically to be used on a student group project, by the students and an evaluation by these students.

Design for Student Investigation

Participants

The participants of the initial study and the creators of the code bases used are students at the University of Canterbury. Following is a description of these students and the related course they are enrolled in for context. The course is a full year paper and is taken by either Software Engineering or Computer Science students in their third year, of three to four years, of university. In the examined year, the students formed into six groups of seven or eight members, which work within a scrum based context to perform all the steps in building a piece of software according to a backlog. This backlog is prioritised by a product owner and guidance of students is provided by the scrum masters. The ceremonies that the students will experience are, stand-ups, planning meetings, reviews and retrospectives. All of the teams work from the same backlog, however, some teams may wish to discuss and implement slight changes, with their product owners input. All groups, separate from each other, will estimate their stories, then based upon this commits a number to their sprint backlog, and from their they break these stories into tasks.

There are no formal lectures for this course, however, there are relevant workshops to help understanding of group related activities. Instead students will be using their past knowledge from both courses completed in past years and from any summer internships that the students have been a part of.

The vast majority of the code written by the students is in Java. As a repository for this code the students use the University of Canterbury's Gitlab server. An example of the tools they are required to use are Maven and some form of continuous integration. The students will use a tool such a Cucumber for acceptance testing. For communication the students are obligatory to maintain a group chat though a tool such as Slack. To complete the project management in an Agile way the students track their projects through a project management tool named Agilefant. This tool provides the ability for the students to log work on their project. This is done in increments against tasks which will then automatically changes their burndown and burnup charts to reflect the work achieved.

The teaching of this course work though very close guidance from staff to the students. All students are marked individually even though there is a shared codebase and group progress. This is possible through a combination of students providing self-reflection, where the students looked at their own performance and the work that they provided to the project over the sprint time period and how they could have positively changed their work, and peer-feedback, where the students evaluate their team members of the quality, quantity and their contributions towards the project during the sprint. At the end of each sprint the students, in each of their groups, present the progress and changes at the review meeting, which also helps to practice their presentation skills. This year there were six teams with a total of 43 students. With the high management the workload for the staff was extremely high and providing immediate, accurate and continual feedback on the codebase and on the constantly changing design of the project is too time consuming alongside other duties.

In this project we look to provide these student groups with tools that gives an abstracted representation of their codebase which allows and promotes the students to reflect on the work and decisions made on the codebase. These tools would not require any overhead for the staff or for the students themselves. A model that encourages the students to reflect is an open student model in the software engineering space. However, as above we do not know how to measure the domain, and no simple rules can be applied to this area, we do not have a set path to build a model for the groups to compare against.

To create this model, we have a rare set of circumstances that help us to make comparisons and situations for the students to make comparisons. The situation is, as mentioned above, the shared backlog which means students are all performing the same tasks as groups. Normally we do not find student groups working on a large scale project which has the same work log.

Using this fact, we are able to use different tools and techniques to help make comparisons between the groups. The students should all have an understanding of their own code base and from this a very approximate knowledge of the process and steps other teams have taken.

The Research Question

The course the students are taking, as stated, is a practical software engineering project course, and creating a model for this is difficult. We do not know which metric or which rules should be applied to such a domain to effectively model it. And we do not know how to visualise any choices we would make. Therefore, in a first step we wish to provide some simple information to groups to allow for reflection on their own codebases and to then provide guidelines to build a more robust system from.

This forms our step by step approach and provides the main research questions.

Following a modelling process, are we able to promote a student's reflection on a codebase through use of comparative visualisations of other groups codebases?

Are we able to use the feedback from the students to create a more robust visualisation system?

Method for Student Investigation

Method Background

Educational systems which are intelligent and/or responsive normally take many experts a long time to build. So before investing a large amount of time and money into designing and building such a system the developers prototype or try to mimic the behaviour of the part they are trying to evaluate. They then use this copy to test to see if the outcomes are as expected. An effective method of building such systems is the layered evaluation [18, 19], where we take each layer step by step, and build each layer using separate evaluation and process to give proof of each step of the process. Then using the results from the evaluation, the actual system is designed and implemented according to the results and processes.

As we do not know the process to build our model or what to use to evaluate our model on we chose to build step by step. This will involve a slow systematic design, implement, trial and then evaluation of each step in the process.

From [20, 21] we can see that metrics are used in a professional environment and can be used to improve efficiency in working in a software environment [22], and in particular that Object

Oriented (OO) metrics can aid in software design and visualisations can elevate their usefulness [23]. Using this knowledge, we selected object oriented metrics as the model on which the visualisations would be built from.

We are confident that the use of metrics in the context of teaching software engineering, and providing valuable information for this purpose. This is because we can see that they can provide information about the state of a software project [24, 25] which can be then used as a platform for evaluation and for students to base comparisons from to achieve reflection.

The Chidamber & Kemerer metrics suite was chosen [26] (See table 1 for descriptions) as these are a very common object oriented metrics suite and mainly because the students already had knowledge of these metrics. This knowledge was through a partnering University of Canterbury Software engineering course that the students were required to take alongside the group course in the first semester of their third year. Chidamber and Kemerer metrics have been extensively evaluated and used throughout the years to describe software projects in an abstract way, and we are confident that these will provide a basis for the model which the students will compare from [27].

We initially use a simple visualisation, the frequency histogram, again for similar reasons, as it is a very well-known model and without much prior knowledge students will be able to identify how the data is being modelled. In this initial case the histogram conveyed enough information to then evaluate student responses and base further systems from. A simple histogram also reduces the time investment into a specific system implementation without having knowledge if the system meets our goals based on the open student model. As stated above this is the initial step in a stepwise process.

Table 1: A table of the Chidamber and Kemerer metrics suite and a small description of each metric included.

Metric	Description
Weight method per class (WMC)	WMC is the implemented method count for a class.
Depth of inheritance tree (DIT)	DIT is the maximum inheritance pathway from the class to the base class in the inheritance configuration of the system.
Number of children (NOC)	NOC is the number of immediate subclasses or children of a given parent class in the class hierarchy.
Response for a class (RFC)	RFC is the number of potential methods executed in response to a message received by an object of that class.
Coupling between object class (CBO)	CBO is the degree of dependency between two classes. Coupling occurs when a class has a call to or accesses another classes variables or methods.
Lack of cohesion methods (LCOM)	LCOM is a measure for the number of unconnected method pairings within a class, and represents the interrelatedness between methods in a class.

Implementation

A set of frequency histograms was implemented for each of the six teams, this set had one histogram per Chidamber and Kemerer metric. To fetch the groups data, each of their git repositories was accessed at the same time period and the master branch was downloaded. We use the master branch to keep the work amount similar over all of the groups and as we would see the most recent stable builds here. At the time of downloading the master branch, all of the groups codebase sizes were under 19,000 lines of code. The amount of code each group had here is still small enough that most members of the group will have an understanding of all of the sections of their codebase and be able to relate back to points on the histograms.

The groups code bases were analysed using an IntelliJ plugin MetricsReloaded [28]. This plugin was used to calculate the Chidamber and Kemerer metrics as there was no other tool practically useable for this purpose. I found here that a large number of tools to calculate these specific metrics have become deprecated with the addition of new version of Java, or the tool would incur a large cost to purchase. Using this plugin caused manual loading of data and manual exporting Chidamber and Kemerer metrics. This exported data was then parsed using a Python 3 script to prepare the data for graphing, which involved correction of null data points and format changing. This data was handed to a statistical tool, The R Project [29], which was able to construct the frequency histograms based on my given parameters and save these to image format. This parsing and creation of histograms can be easily replicated when given a data set.

In the data set we included all of the classes in the project. This did include testing classes, which was intended as students are still able to compare across testing classes. If they were not included a group which focused more heavily on this aspect may not be properly represented in the model and comparisons made by the students may not be accurate.

The histograms were created for ease of comparison across metric type. To achieve this the scale of the histograms was exactly the same across each type of metric. The x-axis always represented the frequency of classes in the histograms bin, and the y-axis always represented the value of the given metric. The use case for having comparisons across one metric type was for comparisons across groups. Allowing the students to directly see their abstract representation of their project against all other groups would allow the students to have an idea based on the given metrics where their group stands.

The setting for conveying the frequency histograms to the students was a web based implementation. An initial page contained a set of selectors, one for each group. Selecting a group would show the chosen groups frequency histogram set. Doubling clicking on a specific histogram moves to a page where we see all of one metric type histogram. This functionality was achieved through simple repeatable JavaScript. There was also a page with the raw Chidamber and Kemerer metrics, formatted from largest to smallest metric values, available to view through direct navigation.

These visualisations created are not novel and can be created quickly, however to achieve good models and results we do not need to have novel creations as we can use many proven techniques [30-32] to achieve our goals. However, the portion where we use these visualisations to make comparisons based upon using the Open social student model technique on a project course with multiple similar groups is novel.

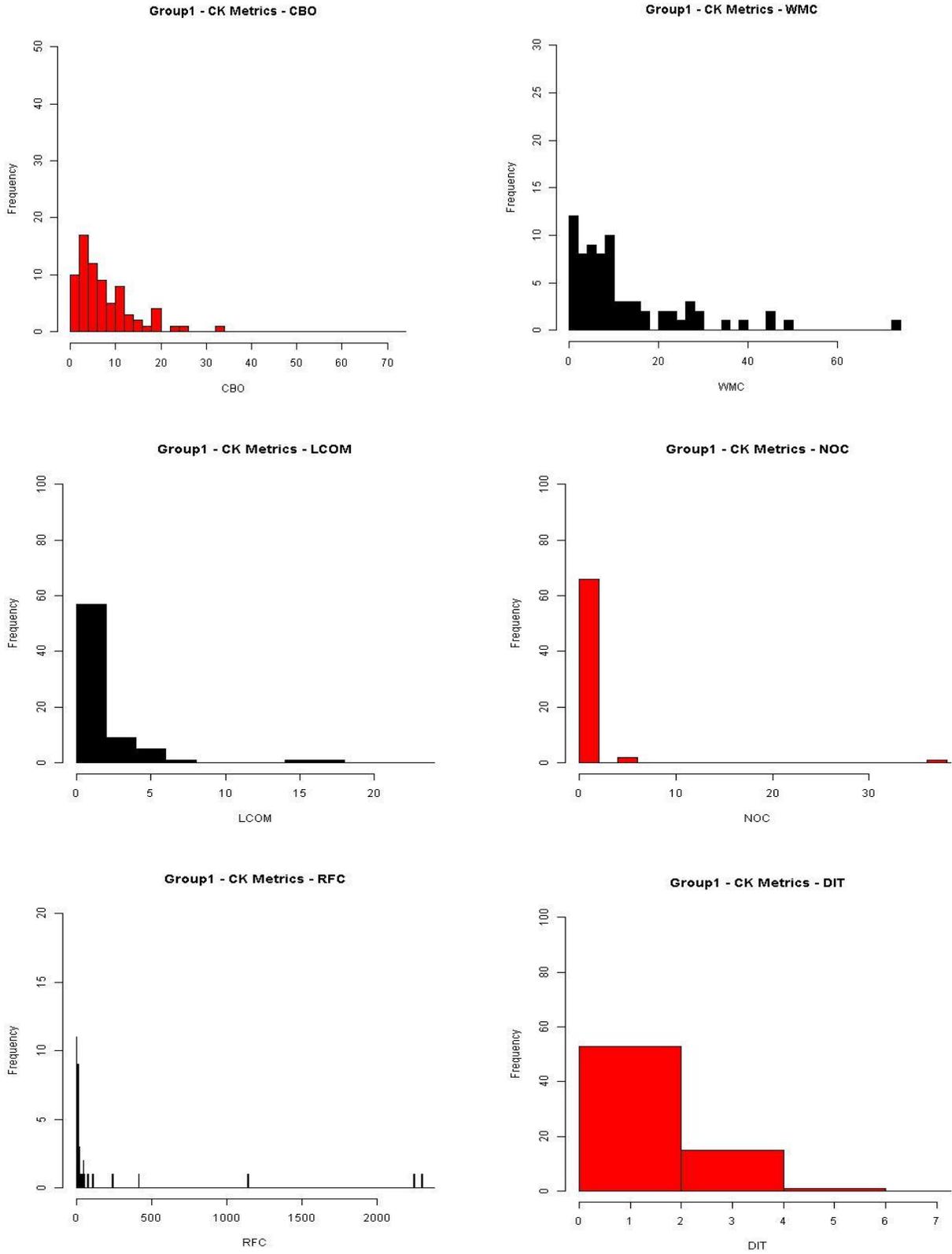


Figure 1. Histograms for a single team showing the Chidamber and Kemerer metrics.

Student Sessions

Each group of students (7-8) was invited to a single discussion each; therefore, there were six sessions. These sessions were scheduled and held on the twelfth week of the student's projects, and was soon after the frequency histograms were made. We did this so the data would be up to date when the students viewed their histograms, meaning the students did not have to recall old information.

Participation to these sessions was voluntary and no monetary or other reward was given to the students. We also made it very clear to the student that these sessions, the discussion or the attendance, in no way would affect the marking of the course or the students' grades. Staff did not advertise or participate in these sessions, as to not influence student and again to make it clear there was no effect on the grading of the course.

The description of these sessions given to students, when they were being scheduled, was that the students would be viewing some simple metrics based upon their codebase. The facilitator had a number of questions on hand to ask the students if discussion was slow or non-existent (see table 2.). However, the main source of output, in the sessions, was from the discussion between students and how they interacted. This form of discussion produced a less structured but deeper output in comparison to filling in a questionnaire. A prime reason for this is the students made comments during the discussions on areas we had not considered to discuss, and therefore expanded our output size.

We divided the sessions into two largely distinct sections. The students were not informed that there was going to be more information provided in the second section of the discussion over what they had already been told. Overall the sessions lasted approximately 30 minutes with the first section being around 10 minutes and the second being around 20 minutes. A very small end of session wrap-up after both sections were complete was done and a follow up to the students a few weeks after the sessions was informally conducted. The students were in a discussion room with the facilitator while all viewing of the content was done on a large screen television connected to a computer which the facilitator controlled.

Section One: A Group Viewing their Own Histograms

In this section, groups were presented with their own six frequency histograms based on the Chidamber and Kemerer metrics (see figure 1 an example of a groups histograms). All of the histograms were presented as the same physical size to the students. This was irrespective of their axis values as these all differed, as the output from different metrics produced different values. The students had the ability to ask the facilitator to interact with the histograms on their behalf. The interactions that could be made were; the ability to view a larger size of the histograms though hovering over a given histogram and the ability to view the underlying data set used to build all of the histograms for a particular group.

In the first section, questions were asked (table 2.) up to the eighth question. I will briefly outline the purpose behind each question. These questions were only asked if the points were not already being discussed or the facilitator wish to clarify responses from the students.

1. How is the project progressing?

This initial question allows the facilitator to have an initial understanding of the group dynamic and the level of progress that the group thinks they have. It is a basic for the facilitator to compare from.

2. Are any metrics being used to track the project?

We know that the students have used metrics in another course, and we look to see if they carried on that use here. This allows us to see if we should expect any of the groups or members to have complete knowledge of what they are about to be shown.

3. Would you want any more information because of these histograms?

This question is asked after the students are shown their own histograms and gives an indication if they are engaged and/or looking to make comparisons based upon what they are seeing in their own histograms. Which in turn is important for the open social student model comparisons.

4. Why would you want this information?

This tries to actively engage the students. We wish to begin classifying the data that students want to be able to see and use and why they wish to see this information on these visualisations. We will then use this to improve the next step of the system.

5. And would this data/information cause you to take any actions in relation to your codebase?

This question allowed us to gauge the effect level that these specific metrics shown in the histograms had on the student projects.

6. Did your group expect the histograms and the metrics to look as they do?

This makes the students do a simple comparison using knowledge they should be familiar with (their codebase) and some abstract representation of the code base and either confirm what they already knew, or if there were points, specifically outliers, that were unknown to the students.

7. How do you think your histograms would compare to other groups?

We then ask the students, without any specific knowledge of how other groups have implemented their projects and therefore, no knowledge of the abstract representations using these histograms, to give a comparative statement about their groups level against the other groups in the class.

8. Would you like to be able to compare to other groups?

Based upon the last question we can then gauge the level of interest in comparison against other group versus the level of interest they have shown in the first section comparing their abstract view of the codebase against their actual view of their codebase.

After these questions had either been asked or the topic had been covered naturally in the student discussions the facilitator would move the session into the second section.

Section Two: Viewing other Groups' Histograms

This section was not known to the students before they were being presented it. In this section each group was shown all of the histograms for all of the other groups (see Figure 2 for an example of the WMC metric shown for just two of the teams). They were presented as six histograms of the same metric type at once, one metric type at a time. As in the first section students could view a larger size of each of the histograms, however, a group could not view the underlying data for any other groups apart from their own. This was because each group had control over class/method names for their project and seeing the names of these in other groups applications could have affected the projects. And so seeing the other groups histograms did not give any more information on these features. We found it acceptable to show the histograms

themselves as these were highly abstract representations and any conclusions drawn from these could not be proven to be correct without more information.

Again as in the first section, questions were asked of the students. (see questions 9-11 in table two). We describe the reasoning.

9. After viewing the other groups histograms, do you think you would make any changes?

This lets us know if the students are comparing their own abstract view to the other groups, taking meaning from the differences and applying it back to their own project. It also tells us if the information conveyed by the histograms is enough to warrant changes being made by the groups.

10. Did viewing the other teams help your group?

This tells us if the experience for the teams was positive and if any of the experiences from the session will help the groups as a whole or help the students individually.

End of Session Wrap Up

After the larger discussion we ended the overall session with a couple of questions about changes and/or improvements to the experience or the visualisations being shown.

11. Are there any different metrics or statistics that you have wanted to be included?

The answers here are then used to assess options for the next step and how to build a system of visualisations. We take into account their answers as they are the test cases.

12. Are there any comments about the sessions as a whole?

Looking for general comments on aspects where we could improve or anything they think that we could have missed in building the visualisations or presenting the visualisations.

Follow Up to the Sessions

An informal follow up would be done, with any of the students in the groups, after a period of 2 weeks. This was to review the answers given to the questions in the discussion session and check if any actions had been taken based upon the discussions.

Table 2: A table of questions wish were planned to be asked to students during the sessions.

Discussion Questions

1. How is the project progressing?
 2. Are any metrics being used to track the project?
 3. Would you want any more information because of these histograms?
 4. Why would you want this information?
 5. And would this data/information cause you to take any actions in relation to your codebase?
 6. Did your group expect the histograms and the metrics to look as they do?
 7. How do you think your histograms would compare to other groups?
 8. Would you like to be able to compare to other groups?
 9. After viewing the other groups histograms, do you think you would make any changes?
 10. Did viewing the other teams help your group?
 11. Are there any different metrics or statistics that you have wanted to be included?
 12. Are there any comments about the sessions as a whole?
-

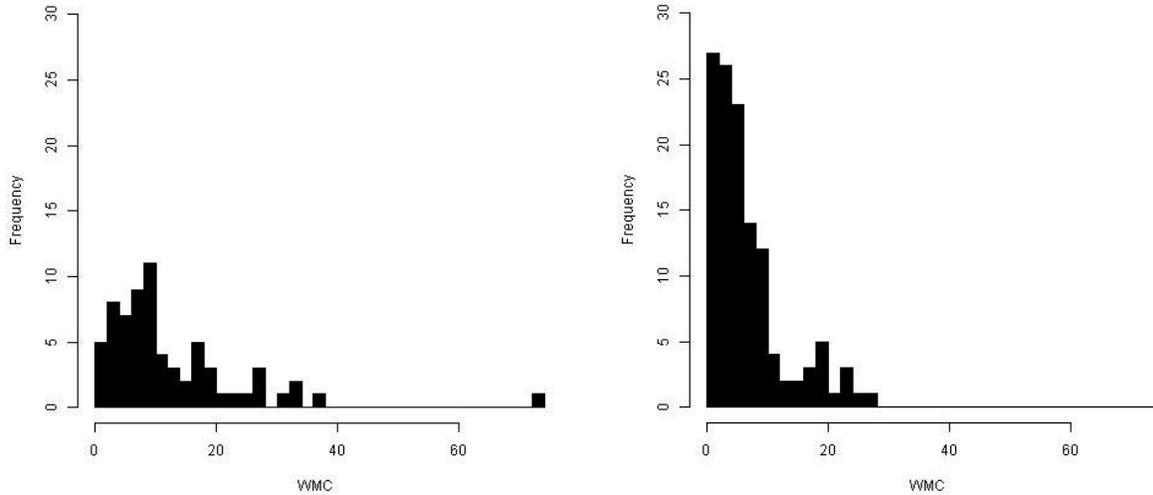


Figure 2. Histograms for two different teams for the Weighted Methods per Class (WMC)

Observations from Student Discussions

The groups were met one at a time across a one-day period. The majority of group members attended with only a couple of members missing in one group. All of the observations made during the session were recorded and a high level summary is given in table 3.

The discussions were led by the facilitator but their input was kept to a minimum. This worked well however, there were specific instances in which the facilitator had inject a question to continue the discussion. Generally, the open ended questions here did not receive good answers, and instead the more directed questions allowed the students to focus and led to better discussion.

Discussion Section One: A Group Viewing their Own Histograms

I note and describe the points made by groups.

All of the groups asked the facilitator to explain the metrics and how they effected the projects. This was understandable as students may not remember these specific metrics or what they meant. And along the same line, the students asked about metric threshold values for these specific metrics. The rationale behind this was seeing how their project ranked against what the many in literature recommend [27]. However, thresholds are not perfect measurements and are

extremely context related. And in certain cases they had seen metrics which were calculated incorrectly, possibly containing imported libraries or Java's SDK.

All of the groups asked the facilitator to bring up the larger versions of the histograms. This was to help in the understanding of what the metrics actually meant in the context of their codebase by trying to find any links between different histograms and therefore metrics. This move towards understanding context led the groups to a main focus on the shape of the graphs and the outliers. This data was found to be interesting to the majority of members over the groups ($\approx 80\%$ of group members), and specifically for the reasons above. However, the groups needed concrete data, such as the names of classes which were offending outliers, to make consider making changes. This led the groups to discussing their project implementations, and which classes and/or method which could be the outliers or causing the histograms shape, before then asking for more detail about the data. We saw that in almost all cases the provided data confirmed what the groups already knew about their codebase.

During this first section, all of the groups were highly interested in seeing other groups histograms primarily for the reasons of comparison.

Some groups made a request to have a description of what being an outlier meant in terms of their project provided on the web page.

Discussion Section Two: Viewing other Groups' Histograms

In this section the average time spent and the focus, viewing and comparing their data with the other teams' data, was significantly higher than in the first section. The histograms in this section were not anonymised as the number of groups was so low.

All of the groups made comparisons between their histograms and the other groups' histograms and gave potential reasons for the differences they were able to find. These comparisons were made at a deeper level than just the metrics, showing more thought and reflection about design and implementation. Their reasons were, sometimes made at an implementation level to explain the differences, however these were guesses as the other groups code was not provided to other groups, and we cannot find out implementation level detail from the histograms. And so any interpretations were done at a high abstraction level with relation to the design.

Five of the six groups asked, and so would have liked, to have more details on the other groups code implementation and four groups wish for specific detail about the design of the other groups projects. Their requests here focused on specific points on the histograms such as high points or outliers which they wished to compare directly to understand the reasons behind. These requests show a performance approach to their comparisons and displayed signs of competitiveness.

To our surprise, five of the six teams had specific reasons for the differences and in their terms effectively explained away differences in the histograms. We can attribute this to the teams only being able to look at a high level abstraction of the code and/or that the groups had made informed decisions for their design choices already. The groups concluded here that their design and implementation was better or at least no different than the other groups.

As the groups believed their position was no worse than other we only saw two teams commit to action items, which were to check and correct certain areas of their code (see table 3 or numbers).

Discussion Session: End

After the two sections of the discussion were completed the facilitator asked for feedback and extra questions regarding the session and the provided histograms. All of the groups wished to be able to continually see updated versions of other groups' visualisations to compare against theirs. We can see that this is performance-oriented as they wished to see if they were better than other groups.

Specific requests that all of the groups made were for: timeline charts and the ability to see and compare to historical states of theirs and others projects, a different range of visualisations which allows them to compare data in a different way, visualisations that incorporated different metrics and lastly the ability to zoom in to see finer details.

Discussion Session: Two Week Follow Up

After two weeks, each group was asked if they had look at any areas of code or made any changes to their code base because of the discussions. Half of the groups had made inspections to their code base while two of these had also made changes directly because of what they had

talked about in the discussions (See table 3). All of the groups were still very keen to see updated versions of the visualisations especially from the other groups.

Four of the groups mentioned that they had tried to find out how the other groups had designed their projects and two of these groups attended other groups' reviews to help them understand the design and the functionality improvements made by these other groups.

Table 3: A table of questions during each session and the response from the teams (Y indicates a yes response).

Student Questions	Group Number						Totals
	1	2	3	4	5	6	
Asked for more information about the histograms.	Y	Y	Y	Y	Y	Y	6/6
Will take actions after seeing these histograms.	-	Y	Y	-	Y	-	3/6
The team expected their metrics to look this way.	Y	-	Y	Y	Y	Y	5/6
Wanted to compare with other teams.	Y	Y	Y	Y	Y	Y	6/6
Would change anything after seeing the other teams.	-	Y	Y	-	-	-	2/6
Would like a greater range of visualisations and information shown.	Y	Y	Y	Y	Y	Y	6/6
Would like to see this information over time.	Y	Y	Y	Y	Y	Y	6/6

Discussion

The reasons only the minority of groups made changes and only half the groups had any actions, was firstly because of the nature of the course the students were taking and secondly the information the information shown that metrics were able to tell the teams.

The course the students are working on is focused on implementation of software, this means for groups to do refactoring they would have to be able to justify it with quantitative data on the time it would save/improvements it would make. The information from these metrics does not provide any concrete data which could be use by the students for this justification, and may only point them in the direction of where to refactor.

The change of code by two teams because of the discussion and the data shown, tells us that the system did have some use, but I would speculate that these changes also had many larger influences that the team already knew about, such as bad architectural decisions relating to that class.

The C&K metrics that the teams were shown at this time may not have been suitable as they only look at classes, which at this point in the projects would be small and well known by all team members. This means that any indications of problems picked up by C&K metrics were generally known about by the students.

The outliers and shape from the histograms were found to be the most interesting points from these. The overall discussion raised some good points about the project in general, and gave the students some insight into how their code base compared against others.

We used an open social student model technique to present groups of students with visualisations of metrics representing their codebase. Showing the students their own histograms provoked a small discussion. We look at this step as not being novel or interesting, however, it the basis for comparison when student viewed other groups' visualisations. Seeing the other groups' histograms in comparison to theirs provoked a deep discussion, which allowed the students to reflect on their techniques and decisions during the project. This is what is predicted from open social student model research and from it we can see that these techniques are useful to increase the reflection engagement and competitiveness in the software engineering project course.

These were exploratory sessions and the findings presented above were used to build the next step in the system which was building an improved visualisation system.

Changes to the System

The system was found to be inadequate for conveying actionable information to the students.

Additions to solve this are:

Table 4: Additions to the visualisation system based upon feedback from student discussion sessions.

Addition	Description
Addition of more visualisations	Initially only one form of visualisation was used (the histogram). However, to convey a greater range of information we will add multiple visualisations. These are the Radar Chart, A time period Line Chart and a modified Pie Chart.
Interactivity with the visualisations	We will add the ability to view specific details of charts and to be able to interact with various aspects of the new visualisations.
The addition of time into the visualisations.	Both the time period line chart and the modified pie chart will use time as an input to be able to compare data over the course of a project and be notified of changes.
Exploring data segmentation	Initially we used the entirety of the project to build our visualisations on. However, in this step segmenting the data based upon other variables, such as time, testing or by package, will be explored.
Quick response visualisations	We can look to provide the user with automatic results based on their data such as alerts on large changes in metrics and providing the students with the visualisations when a regular build is run.

The additions outlined above will be used as the entry point for building the next step in the visualisation system.

Design for Visualisation System

Background on the Visualisation System

The visualisations created in this section are designed to deal with the imperfections of the histograms, given to the students, and the design additions requested by students. We look at only adding feasible and relevant solutions.

The new visualisation suite is to be comprised of three separate yet integrated visualisations. There are multiple solutions to the problems raised from the first step of this project, mainly because solving the problems in one visualisation would create a larger set of problems again. These would be due to information overload mainly. This suite is specifically based on helping student in a group project with reflection on their own code bases and self-evaluation of their knowledge and skills, however, we can apply these visualisations to other projects and still achieve reflection.

We mainly look to address issues with interaction of visualisations, which is achieved in all three, addition of historical data and current data, which we look at in the line chart and the pie chart. The radar chart allows for ease of comparison to achieve a smaller resolution level when comparing data.

Implementations

To create the new visualisation system we use JavaScript and a web container. Again web is used as this allows for the greatest ease of distribution while having access to many frameworks to build visualisation in.

Historical Multiple Line Chart

We can see the result of the implementation below (figure 3.), which is showing the CBO metric for one group over a one-and-a-half-month period. The data is taken on a weekly basis, which was frequent enough for historical data and given the relatively small project size. In figure 3 we only see a small portion of the visualisation, the data goes until early August, however for viewing purposes we cannot fit this on the page and there is only a carry on of data shown.

The data shown in the multiple line chart will be each of the groups Chidamber and Kemerer metrics, displayed one metric per chart. The data is taken from the beginning of the student projects until the current time.

The user is able to interact with this visualisation. The useable functions are comprised of, hovering over a specific class, which selects all of the points the class has and highlights the line between these points. We can select classes for use in the Radar Chart, if we click on a class, or multiple classes and chose the compare button we will be sent to a Radar Chart containing the dated classes data displayed on the Radar Chart. These selections can also be cleared through the clear button.

From this line chart we are able to view all of the classes in a project and see the change of these visualisations over time. This allows for students to perform comparisons from a historical point of view and should encourage reflection on their own codebase.

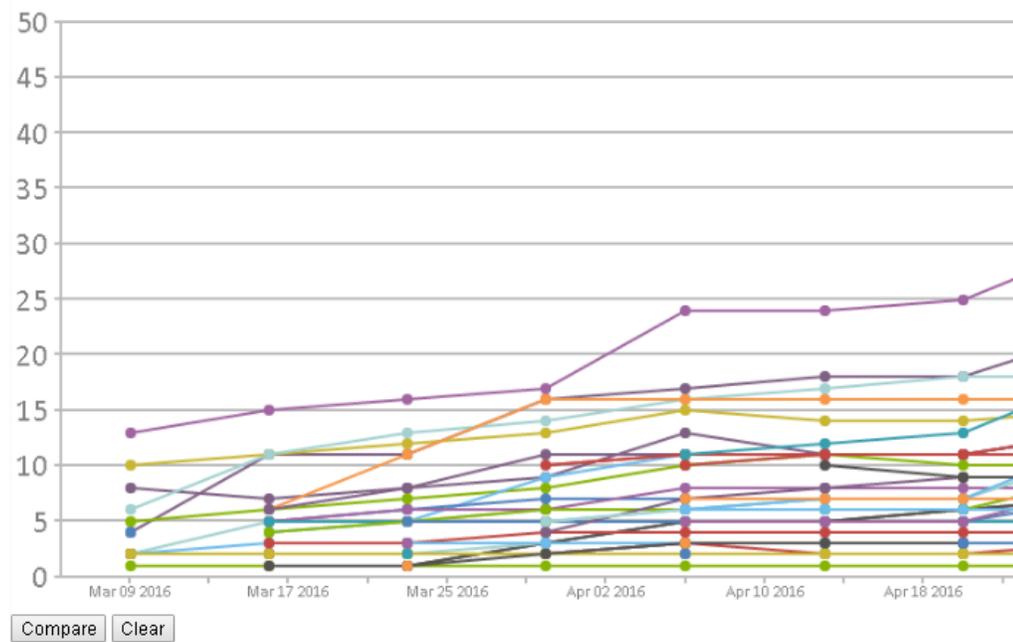


Figure 3. A portion of the multiple line chart showing the CBO metric and historical values.

Radar Chart for Comparison

The radar chart is a visualisation primarily for comparing two objects. It can be used to show multiple points of information from an object in comparison to another object. In our case the points of information are the Chidamber and Kemerer metrics and the objects are classes at a specific date.

As stated above the radar chart can be accessed through the line chart, and only will show what is selected. The radar chart is accessible through any of the metrics line charts.

We look to solve drilling down for specific comparisons with this visualisation. Where in the line chart we can see overall movements and trends the Radar Chart is a direct comparison between two or more classes. And most usefully a direct comparison of the same class over multiple time periods. This allows a student to compare against not only their own data but possibly another teams' similar classes, however this would mean the labels would be removed as to not expose the project design.

We can interact with this visualisation by hovering over any specific point to see the underlying data value behind it.

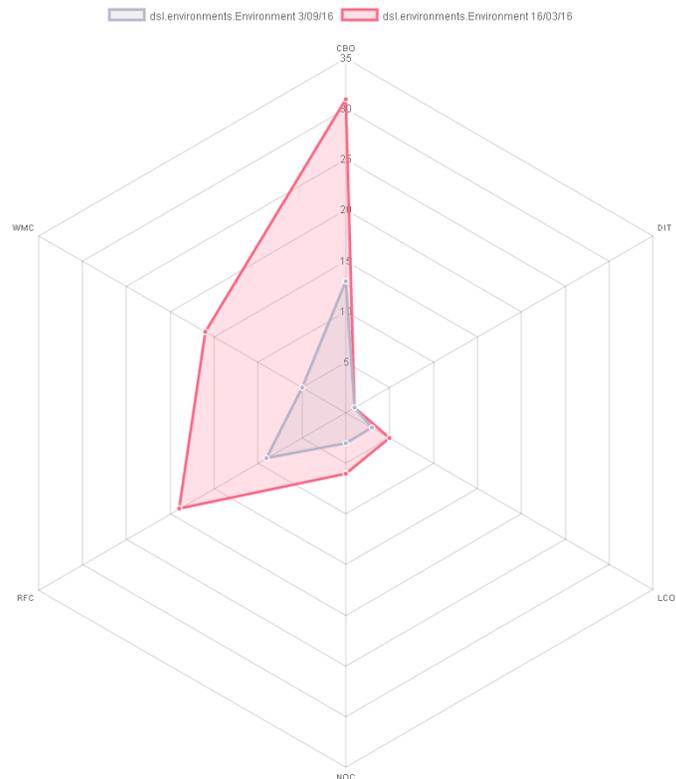


Figure 4. A Radar chart visualisation which is showing a comparison of one class at two different times.

Aster Chart for File Changes

This visualisation looks at the fraction of commits for a particular file over the total number of commits. This visualisation looks to work as a warning system to large unstable file changes in a rapid time period.

This visualisation has 3 areas which can change, these are the colour of each arc segment, the length of an arc segment from the centre towards the perimeter of the chart and the size of the arc in degrees. These are based on the input values of: score, scaled number from 0 to 100 which represents the magnitude of changes, 100 being the greatest amount of changes and 0 being no changes. Calculated by $(\text{additions to class} + \text{deletions from class}) / (\text{max additions} + \text{max deletions}) * 100$. This dictates the arc size from the inner circle to the edge of the arc segment. The weight, a scaled number, which represents the fraction of commits a single class had in comparison to the total of the top 10 classes. Calculated by $\text{commits to a given class} / \text{total commits for the top 10 classes}$. This dictates the percentage of the inner arc segment. And the Colour, which is made from one colour in a gradient range from green to red. Contains 7 steps and is calculated by adding how many CK metric thresholds a class exceeds. We only look at the top 10 classes that have been changed, this is because we want to be able to distinguish each part of the chart easily, and that we would normally expect in a project, like the case study, to have a focus of a few classes, making them take up a majority of the commits. The value in the centre of the image is the percentage of the commits that the top 10 classes make up.

If a class looks like it needs to be looked at because of the size and colour of the arc segment then a user can interact with the visualisation bringing up a tooltip which will tell them the name and path of the class, we do not have this constantly overlaid as it allows for better viewing and does not influence a user's opinion of classes before they examine further. We can also see the magnitude of changes.

This chart is designed as a constantly updating ambient visualisation in a general workspace. The reasoning behind this being, we are able to convey a large amount of meaning though a small amount of information, and therefore allow developers to focus on other aspects of development while still being aware of major changes happening in the code base.

The use of both the number of commits for each class and the magnitude of changes to a file allow a user to see if there is a discrepancy between these, normally we would expect one to be big if the other is big. However, if one is big and one is small we may need to examine the reason for either lots of small changes or a small number of big changes.

These aspects of the chart help the student to constantly reflect on their codebase and will warn them if they may be making potentially bad changes to the codebase.

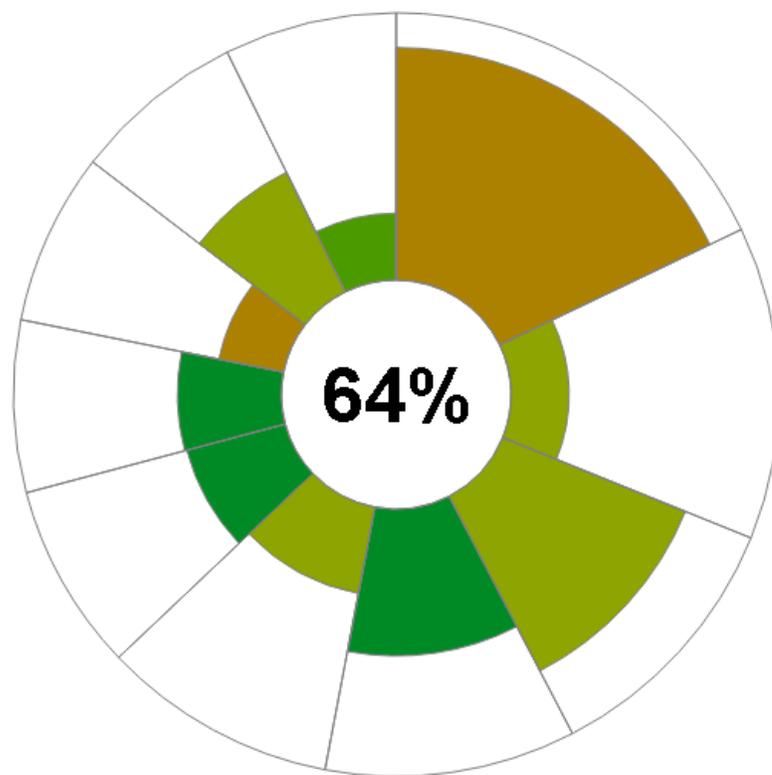


Figure 5: Aster Chart showing commit changes, and code health.

Conclusions

Findings

Using the visualisations to promote student reflection on their codebase was a success. We can see this through the student responses and the action items implemented by some of the groups. Specifically, when the groups viewed their own histograms reflection was observed, but more importantly there was a larger amount of reflection by the students when viewing the other groups histograms. This follows the OSSM predictions in that the students were keen to make comparisons in this area.

However, we have limitations caused through uncontrolled discussion and that when recording the sessions, a notebook was used, meaning parts that were seen as insignificant at the time were not recorded.

We can see that using visualisation that fit more of the criteria that the students outlined after the discussion session would have helped to promote reflection and increase action items by the groups.

These results are important as we are now confident in building a visualisation system for promoting reflection. Therefore, we can spend a greater amount of time building a greater system with more features.

Such a system was proposed and built based off specifications from the student discussions and it is hypothesised to increase student self-reflection.

Future Work

In future we would like to look at having multiple more sessions with the students to continually evaluate the visualisations. This would allow us to collect more reactions on the current step and improve further. This could be achieved with greater ease in a longer term project. Alongside this we would wish to formally validate the second system of visualisations to provide data on their use with relation to OSM and OSSM.

The implementation of automation into the system to allow for consistent publishing to the students would be an additional feature added. We found the problems here are that there is no

automated and consistent metric parser to be used. So if this were to be a new feature we may also have to look at implementing a Java metrics parser which is able to calculate the metrics in the Chidamber and Kemerer metrics suite.

If adding further visualisations into the suite we would like to add a visualisation which can provide soft advice (advice which may be wrong) as it would allow for broader reflection and more general advice. This could look at code smells as they involve soft advice.

References

1. Bull, S. and J. Kay, *Open learner models as drivers for metacognitive processes*, in *International handbook of metacognition and learning technologies*. 2013, Springer. p. 349-365.
2. Bull, S. and J. Kay, *Student Models that Invite the Learner In: The SMILLI:() Open Learner Modelling Framework*. *International Journal of Artificial Intelligence in Education*, 2007. **17**(2): p. 89-120.
3. Kay, J., *Learner know thyself*, in *International Conference on Computers in Education*, Z. Halim, T. Ottomann, and Z. Razak, Editors. 1997: Kuching, Malaysia.
4. Mitrovic, A. and B. Martin, *Evaluating the effects of open student models on learning*. *Adaptive Hypermedia and Adaptive Web-Based Systems*, 2002: p. 296-305.
5. Mitrovic, A. and A. Weerasinghe, *Revisiting Ill-Definedness and the Consequences for ITSs*, in *AIED*. 2009.
6. Kay, J., *The um toolkit for cooperative user modelling*. *User Modeling and User-Adapted Interaction*, 1994. **4**(3): p. 149-196.
7. Uther, J. and J. Kay. *VIUM, a web-based visualisation of large user models*. in *International Conference on User Modeling*. 2003. Springer.
8. Guerra, J., et al., *An Intelligent Interface for Learning Content: Combining an Open Learner Model and Social Comparison to Support Self-Regulated Learning and Engagement*, in *Intelligent User Interfaces (IUI 2016)*. 2016: Sonoma, CA, USA.
9. Elliot, A.J. and K. Murayama, *On the measurement of achievement goals: Critique, illustration, and application*. *Journal of Educational Psychology*, 2008. **100**(3).
10. Beck, K., *Test-driven development: by example*. 2003: Addison-Wesley Professional.
11. Debbarma, M.K.D.S., et al., *A Review and Analysis of Software Complexity Metrics in Structural Testing* *International Journal of Computer and Communication Engineering*, 2013. **2**(2): p. 129-133.
12. Murphy-Hill, E., T. Barik, and A.P. Black, *Interactive ambient visualizations for soft advice*. *Information Visualization*, 2013. **12**(2): p. 107-132.
13. Parnin, C., C. Görg, and O. Nnadi. *A catalogue of lightweight visualizations to support code smell inspection*. in *SOFTVIS 2008 - Proceedings of the 4th ACM Symposium on Software Visualization*. 2008.
14. Mar, L.W. *MAVIS: A Visualization Tool for Cohesion-Based Bad Smell Inspection*. in *Institute of Computer and Communication Engineering, NCKU*. 2011.
15. Telea, A. and D. Auber. *Code flows: Visualizing structural evolution of source code*. in *Computer Graphics Forum*. 2008. Wiley Online Library.
16. Adar, E. and M. Kim. *SoftGUESS: Visualization and exploration of code clones in context*. in *29th International Conference on Software Engineering (ICSE'07)*. 2007. IEEE.

17. Ayewah, N., et al., *Using static analysis to find bugs*. IEEE software, 2008. **25**(5): p. 22-29.
18. Brusilovsky, P., C. Karagiannidis, and D. Sampson, *Layered evaluation of adaptive learning systems*. International Journal of Continuing Engineering Education and Life Long Learning, 2004. **14**(4-5): p. 402-421.
19. Brusilovsky, P., C. Karagiannidis, and D. Sampson. *The benefits of layered evaluation of adaptive applications and services*. in *Empirical Evaluation of Adaptive Systems. Proceedings of workshop at the Eighth International Conference on User Modeling, UM2001*. 2001.
20. Fenton, N. and J. Bieman, *Software metrics: a rigorous and practical approach*. 2014: CRC Press.
21. Mordal, K., et al., *Software quality metrics aggregation in industry*. Journal of Software: Evolution and Process, 2013. **25**(10): p. 1117-1135.
22. Basili, V.R., et al., *Linking software development and business strategy through measurement*. arXiv preprint arXiv:1311.6224, 2013.
23. Harrison, D.R.a.R., *An Overview of Object-Oriented Design Metrics*. 2001.
24. Jureczko, M. and D. Spinellis, *Using object-oriented design metrics to predict software defects*. Models and Methods of System Dependability. Oficyna Wydawnicza Politechniki Wrocławskiej, 2010: p. 69-81.
25. Jalote, P., *An Integrated Approach to Software Engineering*. Texts in Computer Science. 2005: Springer. i-xiv.
26. Chidamber, S.R. and C.F. Kemerer, *A Metrics Suite for Object Oriented Design*. IEEE Transactions on Software Engineering, 1994. **20**(6): p. 476-493.
27. Bakar, A., et al., *Predicting Maintainability of Object-oriented Software Using Metric Threshold*. Information Technology Journal, 2014. **13**(8): p. 1540.
28. Leijdekkers, B., *Metrics Reloaded*. 2016.
29. Team, R.C., *R: A language and environment for statistical computing*. 2016, R Foundation for Statistical Computing.
30. Steele, J. and N.P.N. Iliinsky, *Beautiful visualization*. Vol. 1st. 2010, Beijing; Sebastopol, CA;: O'Reilly.
31. Spence, R., *Information visualization*. 2000, Harlow: Addison-Wesley.
32. Ware, C., *Information visualization: perception for design*. 2000, San Francisco: Morgan Kaufman.