Chen, W. et al. (Eds.) (2016). Proceedings of the 24th International Conference on Computers in Education. India: Asia-Pacific Society for Computers in Education

Towards a Social Student Model in a Software Engineering Group Project Course

Edward ARMSTRONG, Moffat MATHEWS* & Neville CHURCHER
Computer Science & Software Engineering, University of Canterbury
*moffat.mathews@canterbury.ac.nz

Abstract: We describe an observational investigation that explores whether creating open social student models using software metrics promotes reflection. Metric visualisations were shown to six groups in an undergraduate software engineering project course. Seeing their own visualisations prompted reflection on their codebase at a conceptual level as expected; this is as predicted by previous research. When viewing other's visualisations, the focus changed to comparative statements with reasoning to explain differences showing that this increased performance orientation. Delayed queries showed that engagement was varied but increased in all groups.

Keywords: Open Student Model, Open Learner Model, Open Social Student Modelling, User Modelling, Software Engineering Education

1. Introduction

This is the first step towards creating an adaptive system to guide students in a software engineering project course while they are designing and implementing their project. Teaching software engineering group projects is challenging, as there are several ill-defined elements and heuristics with high dependence on context. In our yearlong project course, groups build software against a similar backlog. Each group can view their burndowns/burnups however, due to the ill-natured aspect of the domain, the concept of student/team models does not exist. As there are no set rules for creating "good" software but instead fuzzy heuristics, the modelling of such a domain is very difficult. Furthermore, a software engineering "problem" is often vague and ill-defined. To start this project, we created open social student models for each group using common software metrics and conducted an initial observational investigation.

The purpose of open student models is to give the student the system's view of the student's knowledge or progress within the domain (Mathews, Mitrovic, Lin, Holland, & Churcher, 2012). The hope is that the student builds a high-level view of the domain and increases their meta-cognitive skills helping them become independent learners as they continually reflect on their learning. Recently, work has been done in opening up other's anonymised models to a student to increase motivation and self-assessment skills. Other models (Open Social Student Models – OSSM) might be shown so that the student can compare their knowledge and progress against their peers (Brusilovsky et al., 2016).

In cases like this, it is often recommended that ITS developers build on smaller investigations (even Wizard of Oz studies) as they iterate towards the final ITS; the layered evaluation (Brusilovsky, Karagiannidis, & Sampson, 2004) is an example of this. We started this process from a high-level by creating a visualisation of graphs of one artefact (codebase) using well-known software metrics so that teams could see aspects of their domain without any feedback. After a short period, they were shown metric graphs of other groups. Being an observational investigation, we wrote notes on any interactions, particularly when groups viewed other groups' graphs.

2. Investigation Method

The participants of this investigation were students in a full year project course at the University of Canterbury. Following a method similar to the OSSM, does providing visualisations of other teams' codebase to compare against your own help in reflecting on your codebase?

As metrics are used by professionals to understand their codebase better (Fenton & Bieman, 2014; Mordal et al., 2013) and improve efficiency (Basili et al., 2010), we selected that as our "model" from which to build the representations. We are confident that metrics can provide valuable information for teaching and learning in the software engineering filed as they can provide information about the state of a software project (Jalote, 2012; Jureczko & Spinellis, 2010). We chose the Chidamber & Kemerer metric suite (Chidamber & Kemerer, 1994) as they are common and all our participants are familiar with them as they are taught in a sister theory course. Each of the groups' master branch was downloaded from their git repositories at the same time and frequency histograms were created for each metric. By this stage of the course, each codebase on the master branch only is over 20,000 lines of code. The visualisations do not need to be novel as we can use various proven techniques (Spence, 2000; Steele & Iliinsky, 2010; Ware, 2000). However, seeing if making comparisons using the OSSM technique within a project course is.

Forty-three participants in six groups (7-8 students in each) attended a voluntary session each. The discussion between team members was the focus of the sessions. For the most part, the experimenter simply observed, noted the interactions, and answered any questions.

Each session was split into two parts, the second part was initially unknown to the students. During the first part, each group was presented with their C&K metrics graphs (e.g. frequency histograms) on a large ten-touch screen; this allowed participants to zoom in/out and move visualisations to enable better comparisons. More detail was available for each metric at the granularity of the class or method. In the second part, each group was presented with the graphs for all the other groups. Seeing the other groups' histograms did not give any information about their design or method of implementation.

3. Observational Results and Discussion

All groups asked for a detailed explanation of each metric. This is understandable as metrics are not the students' focus. Metric thresholds were also a point of interest to the student; if these are calculated incorrectly, they could result in spurious results. Understanding the overall picture in terms of their codebase means finding the links between the different metrics. As expected, all groups concentrated a lot more on the shape of the graphs and the outliers. Groups discussed specific classes and what could be affecting the metrics values. In almost all cases, the details confirmed what the team knew about their codebase already. It was interesting to note that spurred on conversations between team members about their code and solutions to a few of the potential problems.

All groups were very keen to see other groups' graphs. They spent a lot more time interacting with the other graphs, including comparing theirs to others'. As there were only six teams, the graphs and data were not anonymised.

All groups had several reasons for any differences between their histograms and other groups'. In this part, it was encouraging to see that the comparisons were made at a much deeper level than just the metric values showing a lot more thought and reflection about their design and implementation. As each group only had access to their own codebase, all references to the possible interpretations of other groups' histograms were at a high level of abstraction in the design. Five of the six groups wanted more details about other groups' code while four also wanted details about their design, showing signs of competitiveness.

Surprisingly, towards the end of the sessions, five of the six teams explained away the differences, settling on the opinion that they were either better than the other teams or at least no different. This could have been because they did not have access to the codebases/designs or because the initial histograms were at a very high-level while the accompanying details were overwhelming, as a few students commented. As such, only two teams had action items to check and correct certain areas of their code.

At the end of the session, all groups asked if, during the year, they could continue to see all visualisations; it was very clear that this was solely related to competitive performance. All students enjoyed using the system and even made requests for change, including timeline charts, a greater number of visualisations, and the ability to change granularity.

Two weeks after the initial session, we conducted a short session with each of the six groups. They were asked if they had taken any actions because of the discussions and to describe what they did.

Three groups inspected areas of their code that they identified as problematic; two of those three groups also made changes. Four of the groups attempted to get the other groups' designs while two of the groups were motivated to attend the other groups' formal sessions to get a better idea of their design and implementations. We could possibly attribute the low number of actions on the codebases due to the nature of the course and the information these particular metrics show. This course emphasises the need of giving the customer value by having concrete results and large refactoring becomes hard to justify within short sprints. The C&K metrics that the teams were shown at this time may not have been suitable as they only look at classes, which in a ~20KLOC project would be a rather small number that is relatively well known by all team members.

4. Conclusions

Open student models have been shown to help students understand the domain better, assess their skills, and plan their own learning. Recent work on open social student models have shown that they can motivate students particularly in terms of performance. Our exploratory observations showed that visualisations instigated group discussion. Students reflected by making connections between the graphs and their work (codebase) and then deciding what, if anything, needed to be done to address it. During this time, they primarily focused on content (such as architecture, code smells, etc.). These observations are in line with previous research into open models. However, when students saw others' models, the focus immediately changed to performance-oriented competitiveness. As predicted by OSSM research, students were very keen to make comparisons. OSSM techniques using metric visualisations seem to be useful to increase reflection, engagement, and competitiveness in the software engineering project course. There are several limitations to this investigation. It was an observational exploratory investigation rather than a controlled study. There were no follow up sessions with new visualisations. The observations were manually recorded. In spite of these limitations, our results show that further investigations in this domain are worthwhile.

References

Basili, V. R., Heidrich, J., Lindvall, M., Münch, J., Regardie, M., Rombach, D., et al. (2010). Linking software development and business strategy through measurement. *IEEE Computer, 43*(4), 57-65.

Brusilovsky, P., Karagiannidis, C., & Sampson, D. (2004). Layered evaluation of adaptive learning systems. *Int. J. Cont. Engineering Education and Lifelong Learning, 14*(4/5).

Brusilovsky, P., Somyurek, S., Guerra, J., Hosseini, R., Zadorozhny, V., & Durlach, P. (2016). Open Social Student Modeling for Personalized Learning. *IEEE Transactions on Emerging Topics in Computing, 4*(3), 450-461.

Chidamber, S. R., & Kemerer, C. F. (1994). A Metrics Suite for Object Oriented Design. [Article]. *IEEE Transactions on Software Engineering, 20*(6), 476-493.

Fenton, N., & Bieman, J. (2014). *Software metrics: a rigorous and practical approach*: CRC Press.

Jalote, P. (2012). *An Integrated Approach to Software Engineering* (3 ed.): Springer.

Jureczko, M., & Spinellis, D. (2010). Using object-oriented design metrics to predict software defects. *Models and Methods of System Dependability. Oficyna Wydawnicza Politechniki Wrocławskiej*, 69-81.

Mathews, M., Mitrovic, A., Lin, B., Holland, J., & Churcher, N. (2012). *Do Your Eyes Give it Away? Using Eye Tracking Data to Understand Students' Attitudes towards Open Student Model Representations*. Paper presented at the 11th International Conference of Intelligent Tutoring Systems.

Mordal, K., Anquetil, N., Laval, J., Serebrenik, A., Vasilescu, B., & Ducasse, S. (2013). Software quality metrics aggregation in industry. *Journal of Software: Evolution and Process, 25*(10), 1117-1135.

Spence, R. (2000). *Information visualization*. Harlow: Addison-Wesley.

Steele, J., & Iliinsky, N. P. N. (2010). *Beautiful visualization* (Vol. 1st). Beijing;Sebastopol, CA;: O'Reilly.

Ware, C. (2000). *Information visualization: perception for design*. San Francisco: Morgan Kaufman.