Continuous assessment in software engineering project course using publicly available data from GitHub

Henrik Gustavsson
University of Skövde
Skövde, Sweden
henrik.gustavsson@his.se

Marcus Brohede
University of Skövde
Skövde, Sweden
marcus.brohede@his.se

ABSTRACT
This paper describes an approach for assessment in a large software engineering project course. We propose an approach for continuously collecting information from a source code repository and collaboration tool, and using this information for assessing student contributions and also for assessing the course as a whole from the teacher's standpoint. We present how we display metrics for how the students perform in relation to some of the requirements of the course. We argue that continuous summative assessment feedback to the students on how they are performing in the project is a suitable strategy for ensuring active participation from the students for the duration of the project course.

Author Keywords
Software Engineering; Education; Project Course; Assessment; Issue management

ACM Classification Keywords
K.3.2 Computer and Information Science Education: Self-Assessment
D.2.8 Software Engineering: Process Metrics

INTRODUCTION
For many years, practical hands-on development work has been seen as a staple for acquiring software engineering knowledge on the undergraduate level [8]. Software engineering has traditionally been taught by using different kinds of project courses [12][13][15]. One way to keep the development environment as realistic as possible is to have large project courses or capstone project courses [15]. Large project courses not only give insights into development but train the students in the collaborative aspects of development as well as providing the technical development skills [9]. Objective assessment in software engineering courses can take many forms [5]. Assessment can make use of many kinds of metrics such as measuring attendance using time sheets [4][5] or by analysis of peer surveys [5] or self-assessment surveys [4]. Metrics can also be collected directly from the software repository [11][14]. Tools and technologies from open source and inner source [3] can be combined with the knowledge learned from large software engineering project courses [15], it is possible to leverage the strengths of both worlds, by allowing the collection of metrics from the publically available platforms and tools.

Given the objectives above, this paper proposes an approach for continuous assessment and self-assessment using data continuously collected from the source code repository and from the issue management tools on GitHub. We detail the automated steps for collecting the data from the publically available repository on GitHub.

First, we highlight the related work, and how our approach advances the state of the art. After that we give a detailed description of the peculiarities of the Software Engineering Project Course that serves as the basis for the approach. Then, we show how the required data can be used by students and teachers for assessment and self-assessment respectively. Finally, we describe the conclusions and propose the future steps of this research.

RELATED WORK
A lot of research has been conducted on the design and evaluation of software engineering project courses [1][2][6][10]. Realism can be achieved by conducting the work as a capstone or large project course [15]. We suggest a novel approach, that in addition to this realism that comes from a large project with many developers, we also work using open source tools and practices, much like in inner source processes [3]. Since all the tools and documentation is completely open, all information in the project can be collected from public information sources such as GitHub.

One key difficulty in assessing student contributions is that the assessment has to take the individual results and the results of the group as a whole into account when assessing the contributions towards the final end product. This becomes particularly difficult when large project courses or capstone project courses are considered [15]. Summative assessment for grading purposes and formative assessment as feedout for students is key for successful project courses [15]. We propose weekly automated data collection and repository analysis as an objective and fair way to achieve both purposes.

OpenSym '19, August 20–22, 2019, Skövde, Sweden
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6319-8/19/08…$15.00
https://doi.org/10.1145/3306446.3340820

Automatic data collection from code repositories has been used for many different tasks such as connecting development logs with corresponding issues [14], analysing developer intentions in comments [7], or to track the number of lines of code in a repository that was written by each developer [11]. It is possible to use the number of lines of code written in the repository as a rough estimate of student participation in the course. However, the number of lines of code does not give any information about work that does not result in code, and it does not differentiate between students that contribute little code but over a long time from students that contribute much code over a short period of time. Conversely, a time sheet [4] gives us a metric for the time spent but if the time spent does not result in any code, it is not trivial to confirm that any work indeed has been performed. Furthermore since this metric is self-reported, this is not necessarily, objective nor necessarily fair since students may overestimate the time spent.

PROJECT COURSE DESCRIPTION

University of Skövde, Sweden is a small university, which in 2015 had 4030 full time students and a teaching staff of 310. The course presented in this paper has a novel organization, that combines a comparatively large project with open source / inner source work processes, and was introduced in 2011 as part of a Web development bachelor program and taught by a team of teachers with backgrounds in software engineering and computer science. The course has for the most part kept the same syllabus each year, and has run every spring since 2011. The data from 2011- 2013 data is however not considered in this paper since either the student population was too small or the students worked on some other artefact. One notable feature of this course is that for the years 2013-2018, each year, the full roster of around 30-40 students each year is divided into a number of groups consisting of around 10 students. Each group works a different aspect of the same artifact, LenaSYS[1], and every year, the students continue working on the same code base, improving the project deliverable produced by the students from the previous year. LenaSYS is a live system, already used in production with actual end-users[2]. The end-users are other students using the system for managing programming and other assignments across a wide range of web-technologies, taught as part of the Web application programming curriculum at the University of Skövde. Working on systems that are in use at the university makes it easier for students to collect requirements and easier for teachers to adjust the project scope [1]. In addition, making source code publicly available would not be in the interest of many companies that are willing to grant students the chance to participate in their projects. Hence, this novel large project course approach expose students to real in-use systems of considerable complexity without having to find companies that are willing to work with or provide publicly available open source code. Open Source / Inner Source [3] ecosystems promote workflows which help student gain knowledge about pull requests into private repositories, resolving conflicts and push changes back to the project on GitHub. Another argument for using LenaSYS as the development artifact in the project course is the fact that the system has been developed over a long period of time and as such carry a significant amount of legacy code which is key for the learning outcomes since it is a more realistic development process [10]. Furthermore, LenaSYS is complex enough to accommodate multiple groups of students looking at different issues, interacting on GitHub, which has lead to a mature and full-featured product that contributes to the motivation of the students.

Year	Total Students	Computer Science	Program Percentage	Web Develop.	Program Percentage
2014	30	10	33%	20	67%
2015	34	13	38%	21	62%
2016	29	9	31%	20	69%
2017	43	22	51%	21	49%
2018	34	18	53%	16	47%
Total	170	72	42%	98	58%

Table 1. Students participation in the project course

Table 1 shows a brief listing of relevant information about students in terms of population size, study program since the course was launched. The project course was offered to students from two distinct study programs: Web applications programming and computer science study program. The computer science program is technically-oriented towards deeper programming and computational concepts, whereas web programmers acquire background in key web knowledge and computer graphics. Both programs include basic programming courses, a Web page design course, a database systems course and a course that introduces software engineering methods. The computer science majors are about a third of the total number of students. Prior to their enrollment into the course, computer science students often communicate that they are not sufficiently knowledgeable in web technologies, whereas web applications programming students worry that they don't have sufficient programming skills. Experience has shown that such worries fade away within the first few days of the project as students start to work better together and complement their skills.

While students would most commonly adopt an iterative agile software development life-cycle [2], our software engineering course resources support this adoption with tools from the open source ecosystem combined with limited traditional instructor-led supervision [1] in order to experience autonomy and self-responsibility in software

[1] https://github.com/HGustavs/LenaSYS
[2] https://dugga.iit.his.se

engineering projects. As students inherit the partly completed end product from previous iterations of the course, students are required to develop a proactive open source management skills, which alleviate issues faced by students when attempting initial contributions to open-source software. The assisted entry into the iterative cycle of open source management will help students to gain progressive competencies leading to new knowledge [11].

By proactive we mean that students must learn to anticipate the flow of events in the project, and be able to work from a running start. When starting from scratch students are not subject to normal maintenance and handling of existing errors. This is quite distinct from repairing errors that each developer recently introduced into code that they are already familiar with, since the code was recently created. It is a much less reactive task to repair errors that were introduced one decade earlier by some other programmer.

DATA COLLECTION

It is key to apply this process each week so that code that is committed but later removed from the main repository can still be tracked. Even though a contribution does not remain in the main repository at the end of the project, the contribution may still be considered code that was contributed to the project. Conversely, code that is written but not merged into the main branch is not collected by the tool. Only code that has been tested and approved by group leaders is thereby counted towards the total for each student.

We have developed a weekly data collection protocol for collection contribution data from GitHub and the source code. This data can then be used by our assessment tool or other tools.

Data collection protocol

The data collection can use the available GitHub API to gather the above mentioned data. A process that uses the API is limited in terms of how many historical items you can retrieve and also how far back the data points resides. This may be a problem for large projects with thousands of commits during a longer interval of interest. In some cases a course can have a duration that is comparable to a whole term or even a full year or longer. The GitHub web site can in those cases be scraped directly without such limitations. Each week during the course, the following information is collected from publicly available information on GitHub.

- The name of each **file** that has been modified, together with information about when the file was harvested from github, and the hash of the commit that contained the update.
- For each file we collect the hash of the corresponding **commit,** the time that the change was committed and the user ID of the student that was the author of that commit.
- For each file we also collect each **span** of lines of code that were modified in that commit, including the full textual content of the modified span.
- For each **student** we collect only the GitHub user id and the year that the student participated in the course
- For each **issue** (or merge) that a student performs, we store the time of the issue, the full text of the issue, and all the information about the **events** that comprise that issue.

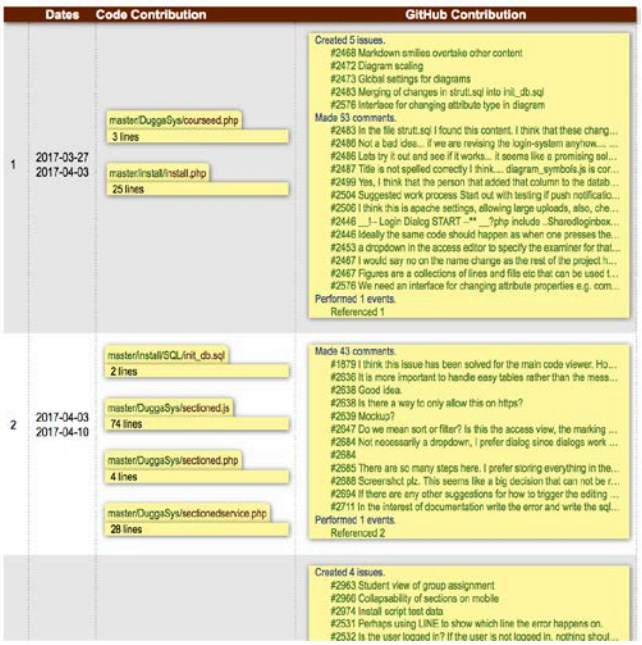

figure 1. Detailed individual contributions

ASSESSMENT TOOL

The purpose of the assessment tool is to provide support for teachers and students in order to assess the individual contribution from each student to the project, in addition to this, the tool provides support for teachers in order to help teachers to gauge the performance in the whole project course. This requires that a teacher or student can monitor the progress of individual students each week of the course and each day in each week.

For teachers, the teacher can select which student data is displayed. For students, the view is locked and can only display the information about the student himself/herself.

The graphs and tables (figure 1,figure 2,figure 3,figure 4 and figure 5) is actual screenshots. Students and teachers see the same graphs/tables. Hence, we have complete transparency in terms of what we use as foundation for our assessment of continuous participation.

The Self Assessment tool does not distinguish between type of code commits. We do consider to add code type separation, e.g., how much LOC is Javascript/PHP/CSS/HTML/Other.

The Self Assessment tool does not categorize students into different levels of experience. We use the tool to gauge participation continuously, but we are less concerned with the level of participation. However, our experience is that students that are not particularly good at programming will excel in other parts of the project, e.g., testing.

At the moment the Self Assessment tool provides the following views:

- A table showing a summary of the total contribution in numbers and rankings (see figure 2). This overview is very similar to the type of information we can show if we only consider the course as a whole [11].
- For each week, a detailed description of the contributions made by the student during the interval (see figure 1)
- a graph showing the contribution per week for the selected student, as a stacked bar chart (see figure 4), a line diagram where students can see their contributions per week day (see figure 3)
- a global ranking table for the collected data (see figure 5) The Ranking is only shown to teachers, and it allows teachers to rapidly identify students that are falling behind on one or more of the metrics. It is key that the list can be sorted dynamically, so teachers can quickly compare the performance of the student in each metric.

Since some of the views are locked to one student at a time, teachers can better assess the detailed contributions from each student. Furthermore, a more complete image can be gained from the rankings or the weekly graph. Graphing individual days and comparing course weeks, allows us to see if contributions are roughly equal over the course and the working week.

Kind	Number	Ranking
Issue Creation	240	1
Comment Creation	285	1
Events Performed	690	2
Lines of Code	147	28
GIT Commit	159	1

figure 2. Table with student's total contribution including a rank for each category

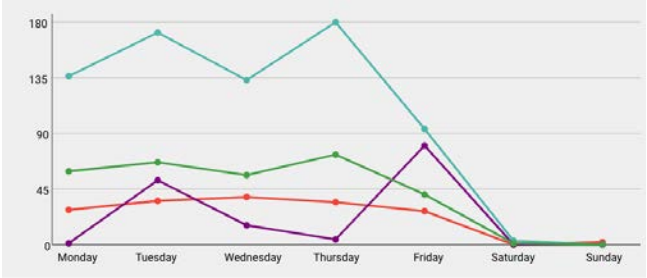

figure 3. A line graph showing the contributions per working day for a specific week or for all weeks summed

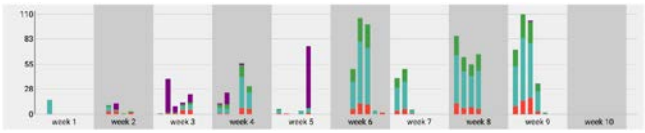

figure 4. A stacked bar graph showing a student's contribution per day/week

	login	alleventranks	allcommentranks	LOC rank	Commit rank
0		10	214	2873	96
1		3	1	2542	57
2		111	61	1057	59
3		252	146	867	92
4		415	130	726	99
5		116	57	675	69
6		10	214	-1	-1
7		3	1	-1	-1
8		2	-1	-1	-1

figure 5. Ranking table for the various metrics (github logins redacted)

CONCLUSIONS

This paper lists three major contributions.

One important contribution is the design of the weekly data collection protocol. This protocol will facilitate continuous objective assessment of student participation in a project course. We conclude that a weekly interval is suitable for this type of large 10-week software engineering project course.

The Self-assessment tools allow each student to gauge individual progress using objective information collected continuously from the source code each week. The color-coding will provide each student with a glanceable approximate measure of the current progress including the current week.

One major advantage of the tool is to show that students have participated in the project and in what way they have participated. We are not looking to categorize students into high/low performers. Coupled with the knowledge of what type of role the student has (project leader or developer) we can assess the participation. For example, a student with less LOC but a lot of GitHub comments and GitHub events is likely to be a project leader. Another example is if a student has no contributions at all for one week. This would allow us to quickly contact this student and to investigate why there is a lack of participation. In such cases where

there are missing participation from students we also can cross-reference with students' diaries.

We also conclude that the objective information gathered from the repository will help teachers to measure the contribution and the participation of each student. Participation can be assessed using the github events and comments, and the technical contribution through the physical code updates. We argue that this is a solid foundation for fair individual assessment in a group exercise.

We feel that the ability to have a tool that shows hard facts about the actual contribution in a software engineering project is very useful. To the best of our knowledge there is no other tool that encompasses measurements from both teamwork facilitating actions (GitHub comments and GitHub events) and Number of files that have been changed, and the number of lines of code that have been modified.

It is way too common that the assessment of student participation is gauged on LOC produced or in a written report produced near the completion of the project course. The former is objective but fails to take into consideration all the teamwork facilitating work done. The latter will seldom show that the participation has been continuous over the course of the project and it is not objective.

One key insight from this work is that it is possible to make a tool that allows us to assess individual student contributions as well as assessing the course outcome using the same tool. However, we have also concluded that we need a way to track the progress of each project group of students as well as the whole course. We also conclude that it would be very beneficial for both self-assessment and summative assessment if the timesheets were also integrated and displayed alongside the github events.

FUTURE WORK

One important future development would be to provide more detailed grouping / aggregation of information. It is very important to be able to know if there are problems within a group of students rather than within the course as a whole. Currently this information can only partially be deduced from the prior knowledge of the teachers involved in the course. In prior years, individual projects groups have had severe problems that were not visible in the course average information. If we display the data from the view of the issue, or the individual code files, students or teachers could use this software to identify development bottlenecks related to the development process.

Another key future development would be to add a way for students and teachers to view the complete updated content, so that the code could be evaluated rather than knowing that, for example, a number of lines were updated in a certain file. Currently this requires a cumbersome round-trip navigation to github, which slows down the work-flow for the teacher that is performing the assessment.

We also foresee that this type of toolkit could be used to make an analysis of student contributions in other courses that make use of public github repositories, such as final year project courses. In this case we would be interested in monitoring the development process of a single student, to ascertain that the student has continuously been writing program code rather than copying code from other sources. We envision that this type of analysis tool will be helpful for detecting anomalies in repositories such as unallowed cooperation.

The assessment tool views all github events as equal. Some events such as commits could be considered more valuable, and some events such as tagging could be considered less valuable. Furthermore, solving bugs and working on issues resulting in solved bugs should be more valuable than working on orphan issues which do not result in neither commits nor other further work.

It has been previously shown [4] that timesheets play an important role for student assessment; our system collects time sheets, but not in a form that can be connected directly to the data gathered from github. One interesting future work task could be to integrate the time sheets gathered from the students with the data collected from github. This way the time sheets can be used for both assessment and self-assessment alongside the data created from github. Such time sheet data will show us more detailed how the workdays look. Do students start at 8 and finish by 17? Are they more productive in the morning or afternoon?

Students are required to keep a diary during the project. Each day they are supposed to write 2-3 sentences of what they did (or didn't do). We cross-reference with this diary if we see a dip in participation activity. Useful for example if a student has been ill. We have plans to incorporate the diary in the tool as well.

One important consideration is how students react to the assessment tool. How do they feel about the transparency? Do they think that their grading is fair and objective? Etc. We have started to collect student responses through course evaluations, but would like to do a more formal study investigating students' appreciation of the assessment tool.

Finally we propose that the collected information can be used to perform scientific analysis of the progression of the course over the years.

REFERENCES

1. Tero Ahtee. 2003. Inspections and historical data in teaching software engineering project course. In *Proceedings 16th Conference on Software Engineering Education and Training*, 2003. CSEE T 2003. (2003)
2. Maria I. Alfonso and Antonio Botia. 2005. An Iterative and Agile Process Model for Teaching Software Engineering. In *18th Conference on*

Software Engineering Education & Training (CSEET'05), Ottawa, Ont., 2005, pp. 9-16. doi: 10.1109/CSEET.2005.5

3. Noel Carroll, Lorraine Morgan, Kieran Conboy. 2018. *Examining the Impact of Adopting Inner Source Software Practices*. 1-7. 10.1145/3233391.3233530.
4. Nicole Clark, Pamela Davies, and Rebecca Skeers. 2005. Self and peer assessment in software engineering projects. In *Proceedings of the 7th Australasian conference on Computing education - Volume 42* (ACE '05), Alison Young and Denise Tolhurst (Eds.), Vol. 42. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 91-100.
5. Jian Chen, Guoyong Qiu, Liu Yuan, Li Zhang and Gang Lu. 2011. Assessing Teamwork Performance in Software Engineering Education: A Case in a Software Engineering Undergraduate Course. In *2011 18th Asia Pacific Software Engineering Conference*, Ho Chi Minh, Vietnam, December 5-8, 2011. 17-24. 10.1109/APSEC.2011.50.
6. Michael Gnatz, Leonid Kof, Franz Prilmeier, and Tilman Seifert. 2003. A Practical Approach of Teaching Software Engineering. In Proceedings of the 16th Conference on Software Engineering Education and Training (CSEET '03). IEEE Computer Society, Washington, DC, USA, 120– (2003)
7. Emitza Guzman, David Azócar, and Yang Li, Sentiment Analysis of Commit Comments in GitHub: An Empirical Study. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, New York, NY, USA, 2014, pp. 352–355.
8. Jane H. Hayes, Timothy C. Lethbridge and Dan Port. 2003. Evaluating individual contribution toward group software engineering projects. In *25th International Conference on Software Engineering, Proceedings.*, Portland, OR, USA, 2003, pp. 622-627. doi: 10.1109/ICSE.2003.1201246
9. George Mitchell and Declan Delaney. 2004. An assessment strategy to determine learning outcomes in a software engineering Problem-based learning course. *International Journal of Engineering Education*. 20. 494-502.
10. Tom Nurkkala and Stefan Brandle. 2011. Software Studio: Teaching Professional Software Engineering. In P*roceedings of the 42Nd ACM Technical Symposium on Computer Science Education (SIGCSE '11)*. ACM, New York, NY, USA, 153–158.
11. Vedran Ljubovic and Novica Nosovic. 2012. Repository analysis tools in teaching Software Engineering. in *2012 IX International Symposium on Telecommunications* (BIH□), pp. 1–5.
12. Jochen Ludewig and Ivan Bogicevic. 2012. Teaching software engineering with projects. In *Proceedings of the First International Workshop on Software Engineering Education Based on Real-World Experiences* (EduRex '12). IEEE Press, Piscataway, NJ, USA, 25-28.
13. Valentin Razmov. 2007. Effective pedagogical principles and practices in teaching software engineering through projects. In *37th Annual Frontiers In Education Conference - Global Engineering: Knowledge Without Borders, Opportunities Without Passports*, Milwaukee, WI, pp. S4E-21-S4E-26. doi: 10.1109/FIE.2007.4418158
14. Bilyaminu Auwal Romo, Andrea Capiluppi, and Tracy Hall. 2014. Filling the Gaps of Development Logs and Bug Issue Data. In *Proceedings of The International Symposium on Open Collaboration* (OpenSym '14). ACM, New York, NY, USA, Pages 8, 4 pages. DOI: https://doi.org/10.1145/2641580.2641592
15. Maria Vasilevskaya, David Broman, and Kristian Sandahl. 2015. Assessing Large-Project Courses: Model, Activities, and Lessons Learned. *Trans. Comput. Educ.*, vol. 15, no. 4, p. 20:1–20:30, Dec. 2015.