Towards a Supercollaborative Software Engineering MOOC

William Billingsley

NICTA GPO Box 2434, Brisbane Queensland 4001, Australia william.billingsley@nicta.com.au

Jim R. H. Steel

The University of Queensland Building 78, St Lucia Queensland 4072, Australia j.steel@uq.edu.au

ABSTRACT

Recently there has been rapid growth in the number of online courses and venues through which students can learn introductory computer programming. As software engineering education becomes ore prevalent online, online education will need to address how to give students the skills and experience t programming collaboratively on realistic projects. In this paper, we analyse factors affecting how a supercollaborative on-campus software studio course could be adapted as a project-led supercollaborative MOOC.

Categories and Subject Descriptors

K.3.2 [Computers and Education]: Computer and Information

Science Education – computer science education

General Terms Design, Human Factors.

Keywords Software Engineering, Massively Open Online Course, **Studio Course**, Continuous Integration.

1. INTRODUCTION

Recently, there has been rapid growth in the number of online MOOCs (massively open online courses) and venues teaching introductory computer science and programming. Coursera, edX, and Udacity each offer multiple such courses. In 2012, 150,349 students enrolled in **CS50x from Harvard University on edX,** with 1,388 receiving a certificate of completion [1]. Khan Academy, Codecademy, Treehouse, and other sites also offer students support for online study in programming. These courses predominantly teach students to program on their own. BerkeleyX’s Software as a Service course [2] includes collaborative topics and encourages pair programming, but the scale of collaboration students undertake is small. As software engineering education becomes more prevalent online, we suggest that online education will need to address how to give students the skills and experience of programming collaboratively on realistic software projects. This has been an active topic of research for on-campus courses for more than twenty years [3, 4, 5], but is an area that online education is largely yet to address.

In previous work [6, 7] we described an on-campus course centred around a software project that has extensive inter-as well as intra-group collaboration. Students work in small groups to develop a feature for a program, but must coordinate and integrate with many other groups also contributing features. In 2011 and 2012, we had approximately 70 students working on a single codebase; in 2013 this rose to over 140 students. For the purpose of this paper, we refer to this kind of project, in which students collaborate in groups which in turn collaborate with other groups, as a supercollaborative project.

We are adapting our course to offer it as a MOOC. Online students would access videos, tutorials, exercises, and can take part in supercollaborative projects. On-campus students would have the additional resources of lab studio time, and interactive workshops and demonstrations by the lecturers in class.

In this paper, we discuss considerations that impact on the course’s viability online, and how the design will need to change. Until the MOOC runs we will be unable to fully evaluate the outcome, but we present this paper so that the analysis and issues involved may be shared with the software engineering education community.

2. ON-CAMPUS COURSE SUMMARY

The course was developed in 2011, and for the first two iterations was predominantly taken by second-year students in the software engineering and information technology programs. In this version of the course, students collaborated on developing features for a fork of the **Robocode open source project.**

A restructuring of the school’s degree programs caused the class size to more than double in 2013, with the addition of many multimedia design and information systems students. In this iteration of the course, we changed the project to give students a much smaller bespoke starting code base to build from. The project we chose was to build a multiplayer games arcade.

**Code collaboration takes place on GitHub,** supported by continuous integration servers that build each commit pushed to the master branch of the repository. Automated builds, test coverage, and automated software metrics give the class ready access to information about the health of the project.

Alongside the project, a weekly lecture series covers topics on distributed version control, automated builds, continuous integration, debugging, testing, design patterns and other topics relevant to their collaborative project. An unstructured weekly studio session is also provided, giving teams a weekly timeslot when they have access to in-person support in their project work from tutors, the lecturers, and each other.

In 2013, 172 students enrolled at the start of semester, with 146 still enrolled in the final week. Over the semester, the class made 4,883 commits and grew the project to 67,900 lines of code. The students developed included many games as well as cross-cutting features such as a common overlay, achievements, networked multiplayer support, and an event recording/replay facility.

3. IMPACT OF CLASS SIZE AND ATTRITION ON VIABILITY OF SUPERCOLLABORATION

While an on-campus course of 172 can place all the students on the same project, a MOOC that could potentially have orders of magnitude more students will need to subdivide the class into many project cohorts.

MOOCs typically have also a much higher rate of attrition than on-campus courses. The quoted attrition rates can be 90% or more. For example, of 150,349 students registering for **CS50x** only 1,388 (0.9%) received a certificate of completion [1]. For MITx’s first course, 6.001x, of almost 155,000 registering only 7,157 (4.6%) passed the course [8], and Philip Zelikow’s Coursera course on the history of the modern world passed nearly 5,000 (10.6%) of nearly 47,000 students [9].

Most of this attrition occurs early, and for many MOOCs a high attrition rate might not be problematic as students can gain value from a course without completing it. For CS50x, only 10,905 (7%) students submitted the first problem set, 10,137 (7%) indicated they intended to complete all the coursework, and only 3,381 (2.3%) indicated they took the course because of the prospect of the course certificate [1]. For 6.001x, 23,000 students (14.8%) submitted the first problem set, and 9,000 (5.8%) passed the midterm quiz.

However, attrition could have a much more adverse effect in our course. In courses that are taken individually, if a student drops out they are the only one affected. In a team course, students dropping out could cause the team to become unviable, leaving their teammates “stranded”. In a supercollaborative course, if too many teams on a project become unviable, the project may become unviable or ineffective, affecting the other teams. There is a question, then, of whether the high attrition rate of a MOOC would cause a supercollaborative course to become unviable. The projects must be manageable when student numbers are at their peak, but must remain viable and pedagogically effective as participant numbers fall.

If the viability of the projects can be ensured, then the attrition within teams can be mitigated and potentially used to pedagogical advantage. Reallocating stranded members to other teams tests both the arriving member, in comprehension of a new system, and the team, in providing transparency and documentation of the group’s product and processes. In the on-campus course, students undertake a single long sprint, allowing them to encounter all of the course content before a sprint cycle finished. But in an online course, shorter sprints may be preferable, to better reflect real practice and to provide a natural point at which to reallocate students between teams.

3.1 Minimum Pedagogically Effective Size

When we designed the on-campus course, one of our design principles was that students should be faced with problems for which the most effective solution is to use the techniques that we are teaching.

This was one of the reasons for putting multiple groups on the same codebase. On small projects, with only intra-group collaboration, it is feasible for students to coordinate their activity just by talking to each other, without using the processes, discipline, and tools that the course teaches. It is when someone you do not talk to frequently starts modifying your code in unexpected ways that you discover the value of good tests, version control, and an issue tracker. By requiring inter-group collaboration, we scaled the project far beyond what conversation alone could support, and forced students to encounter the problems that the course teaching addresses rather than circumvent them. We dubbed this the "feel the pain" pedagogy.

In a MOOC, students are generally remote from each other, although there will be some groups of students who are colocated. Students can join a MOOC from all around the world, but some students who already know each other might decide to sign up together, and local study groups are often formed.

So long as a project does not consist only of colocated teams, it would be difficult for students to coordinate through conversation alone. Remote communication would be a barrier. This suggests that in a MOOC environment, the minimum size of a project for it to be pedagogically effective could be smaller than in an on-campus environment. In the three iterations of the on-campus course so far, we have always had at least nineteen teams collaborating on a single codebase. Online, a project (a single codebase) could potentially remain pedagogically effective with far fewer.

3.2 Maximum Manageable Size

If attrition is to occur throughout the course, this would suggest that the projects would need to be reasonably large at the beginning so that they do not shrink too small by the end. However, even at its peak size, each project must be manageable.

There are two aspects of the project that we consider may become unmanageable: the code base, and initial feature selection.

With more students and more teams modifying the code, we would expect the code base to change more rapidly. This could make it difficult for some students to keep up with a fast-moving target. However, on campus we have found that the code changes fastest at the end of the project, as teams rush to finish their work before the due date. Figure 1 shows the number of commits per week in the on-campus course in 2013. This end-of-project rush does cause problems for students and is something that we are seeking to reduce. But it is worth noting that in a MOOC it would occur when attrition has already taken place and projects are at their smallest.

Feature selection, where groups decide what they would like to work on, occurs earlier in the course. While groups may implement more than one feature, typically they have one feature that is their main focus, and usually it is the one they first selected.

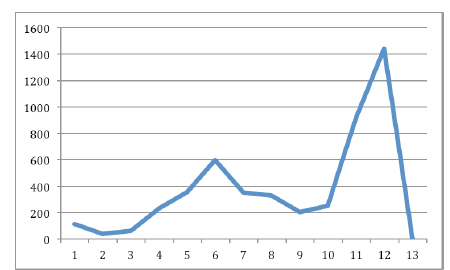


Figure 1: Commits per week in the on-campus course in 2013.

This feature selection drives aspects of both the collaboration and the design. As teams try to ensure that their own work is coherent (so they have a clear vision of what they are trying to achieve) and has loose coupling with other teams (so they do not need to wait on each other’s changes), the teams’ choices of features can be seen as a quickly established high-level design for the system.

In 2013, thirty-one teams collaborated on the project. Fifteen of those teams worked on infrastructure features and APIs that could be consumed by other teams, and nineteen teams developed games. (Three did both.) While nineteen (in 2012) and thirty-one (in 2013) teams succeeded in choosing well-fitting features with a reasonable separation of concerns, that number probably does not scale up indefinitely. At some project size, there would be too many teams in the system for each student to have a reasonable grasp of what the other teams are choosing to do.

In our view, then, it is feature selection rather than code velocity that is likely to impose an upper limit on the viable project size.

3.3 Individual Work as well as Group Work

One strategy for minimizing attrition in the projects would be to begin the project work after much of the course attrition has taken place. This also fits the practical needs of the project structure.

In Figure 1, there are comparatively few commits until week 4. Before students can begin making significant changes to the code, they need to obtain it, understand some part of it, and have a reasonable idea of a change they would like to make. Thus the first part of the course requires learning version control (to get the code), build systems (to get the code working), and ways of exploring and understanding the existing code. This has to happen before collaboration can start taking place. We ask students to form their groups by week 3 of the course, and to start establishing what feature they would like to work on in week 4.

**CS50x** is the MOOC with the lowest nominal completion rate that we have found, so makes for a suitable pessimistic scenario to test against. In CS50x, 3,292 students submitted the problem set in week 3, and 5,259 students completed the subsequent quiz [1]. If we take the higher of those two numbers (rather than course registrations) as our base, then the attrition rate from there on in is 73.6%, rather than 99.1%. This is still very high, but in this scenario supercollaboration would be viable.

Figures 2 and 3 show the mean results of twenty runs of a computer simulation in which students are placed in teams of ten, with fifty teams per project. The simulation repeatedly removes a student from a random team according to the attrition rate. The chart shows the survivability of teams, and the survivability of projects, where 150,000 students are placed into groups and then undergo 99% attrition, and where 5,250 students are placed into groups and undergo 74% attrition. In the former case, half the surviving teams are left with only a single team member, and half the projects are left with only a single team. However, in the latter scenario, the median team size is three and all of the projects in every run had at least fourteen active teams remaining.

The results here are not intended to be predictive. So far we know very little about the factors that affect MOOC attrition rates, and it is unlikely that students would leave groups randomly, but it suggests that even under a reasonably pessimistic scenario, supercollaboration is not obviously doomed to collapse. A lead time of individual work before the project begins can bring the attrition rate during the project below a rate where it is viable.

Automated quizzes and exercises, which are typical to most MOOCs, would also have two additional benefits for the course.

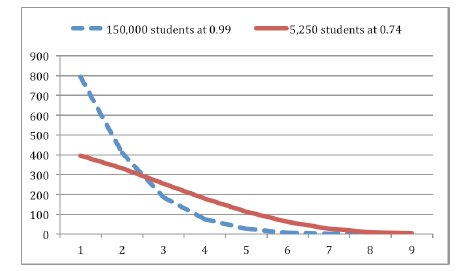


Figure 2: The number of teams with at least n students after attrition. Mean of twenty computer simulations.

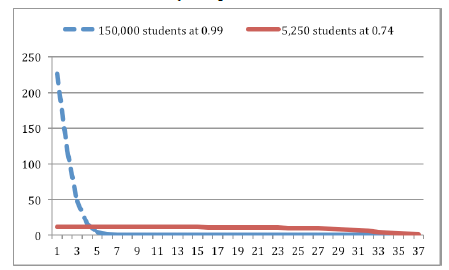


Figure 3: The number of projects with at least n non-empty teams after attrition. Mean of twenty computer simulations.

Firstly, by exposing students’ activity (though not their marks) in individual exercises, the course could encourage active students to cluster together into teams. This would provide some additional defence against attrition, as the harm occurs when an active student finds that his or her collaborators are inactive.

Secondly, it would improve the course’s support for trying out a technique in a tame environment, before applying it to the project. The project compels students to apply some concepts frequently, such as merging changes, and produces immediately visible outcomes for these. However, on-campus we observe that other topics (e.g. refactoring) have longer feedback loops. The teams consuming a code change might not be ready to work with it immediately, and students do not always view the static analysis reports. Automated exercises would ensure students can test their understanding of these topics with immediate feedback.

4. SELECTIVE PARTICIPATION AND HETEROGENEITY

In a traditional on-campus course, the knowledge, background, and interests of the students are heavily constrained. We know that students have taken the prerequisite subjects, and that they have not taken the subsequent subjects for which this is a prerequisite. We know that, as university students, they will have a broad focus across most of the topics we require for the exam, because they need to pass their courses in order to obtain their degrees. In a MOOC, this is not the case. Some students may already have degrees, while others may still be in high school.

In 2013, our on-campus cohort became somewhat more varied as multimedia design and information systems students came into the course alongside the software engineers. To cater for this, we changed the project from improving a pre-existing product (Robocode) to creating a new system from a very small base. As the base was small, it was comparatively unconstrained, and teams were able to specialise. Some teams took it upon themselves to define the design language of the arcade that teams should fit into. Other teams built games. Others built infrastructure that is used entirely behind the scenes. In terms of the balance of teams, we regard this as a successful change.

For a MOOC cohort that is even more varied, the course would need to be more flexible still. Some students might only be interested in particular topics within the course, and have no time for the project. Others could be well versed in the material, but perhaps interested in taking part in a project with students in order to try taking on a more senior role on the team than they have in their workplace. In our design, the supercollaborative project would be the largest component of the course. Teaching content would comprise topic-based sequences of videos and exercises, with each topic sequence able to stand alone, but also forming part of a consistent narrative within the course. The aim would be to ensure that new topic sequences could be added as needed, without the course becoming disjoint.

In a software engineering course like this, there are also many voices other than the course staff that would be of interest to students. There are many professional materials, videos, presentations, and an increasing number of online playgrounds and tutorials available. In previous work, we proposed courses where smart exercises were supported by a dynamic ecosystem of socially discovered, written, and added content [9]. The goal of dynamically evolving courses is still far off, but we can begin by including interviews with practitioners from companies and open source projects, and other materials by people outside the course.

5. PEER MARKING OF GROUPS

In the 2013 course, we introduced a critique at week 8. Groups presented their progress, and sought feedback from their colleagues. Students were allocated five groups to mark and critique by filling in a simple survey. The textual feedback in each critique was then made available to the group being criticised, who would in turn mark the critique on whether it was objective, specific, actionable, constructive, and useful. In total 725 critiques were completed, and 3,075 reviews of critiques.

The only staff intervention we were called to undertake in the critique marking (apart from writing the software to support it) was handling a small number of requests to fix student errors – particularly where students had submitted a critique for one team that they intended to be for another. However, as each group received more than twenty critiques, even if uncorrected these errors would have had negligible impact on the marking.

In our on-campus course, teams’ final project outcomes are marked by the course staff. Within the teams, students use a peer grading system to divide their project mark between them according to their contribution. However, it appears it would be viable to replace the staff marking with a sequence of peer critiques, that might also encourage students not to leave so much work for the final week’s rush.

6. CONCLUSION

Supercollaboration in the classroom is comparatively new. As far as we know, our course is still the only software engineering course to place so many teams of students on a single project. Supercollaboration in the open online classroom, then, is always

likely to be a step into the unknown. However, our analysis suggests that it is feasible, and we are in the process of producing such a course. The key aspects appear to be keeping the elements of the course coherent but independent, and ensuring that project groups only form after the initial high student attrition has occurred, when there is data to identify and group active students.

7. ACKNOWLEDGMENTS

We gratefully acknowledge the assistance of our course tutors, Jackson Gatenby and Phillip Drew. as well as Jorn Guy Su., with whom we co-designed the first iteration of the course. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

8. REFERENCES

[1] Malan, D. 2013. Data, data, data (from CS50x) http://harvardx.harvard.edu/blog/data-data-data-**cs50x** Accessed 23 Oct 2013.

[2] Fox, A. and Patterson, D. 2013. CS169.2x Engineering Software as a Service: An agile approach using cloud computing -MOOC. http://beta.saasbook.info/courses Accessed 24 February 2014

[3] Tomayko, J.E. 1991. Teaching software development in a studio environment. Proceedings of the twenty-second SIGCSE tehcnical symposium on computer science education. 300-303

[4] Docherty, M., Sutton, P., Brereton, M., and Kaplan, S. 2001. An innovative design and studio-based CS degree. ACM SIGCSE Bulletin. 33, 1, 233-237. ACM

[5] Hundhausen, C.D., Narayanan, N.H., and Crosby, M.E. 2008. Exploring studio-based instructional models for computing education. In Proceedings of the 39th SIGCSE technical symposium on Computer science education (SIGCSE '08). ACM, New York, NY, USA, 392-396.

[6] Billingsley, W. and Steel, J. 2013. A comparison of two iterations of a software studio course based on continuous integration. In Proceedings of the 18th ACM conference on Innovation and technology in computer science education (Canterbury, UK, July 01 -03, 2013). ACM, New York, NY, 213-218.

[7] Su., J. G., and Billingsley, W. 2012. Using continuous integration of code and content to teach software engineering with limited resources. In 34th International Conference on Software Engineering (Zurich, Switzerland, June 02 -09, 2012). 1175-1184

[8] Hardesty, L. 2012. Lessons learned from MITx’s prototype course. MIT News, 16 July 2012. http://web.mit.edu/newsoffice/2012/mitx-edx-first-courserecap-0716.html Accessed 23 Oct 2013.

[9] Anderson, N. 2013. U-Va. MOOC finds high attrition, high satisfaction. The Washington Post, 13 May 2013. http://articles.washingtonpost.com/2013-0513/local/39241038\_1\_coursera-massive-open-onlinecourses-moocs Accessed 23 Oct 2013

[10] Billingsley, W. 2008. The intelligent book: technologies for intelligent and adaptive textbooks, focussing on discrete mathematics. Technical report UCAM-CL-TR-719, University of Cambridge

Совместный

**К суперсовместному Mooc программной инженерии**

АННОТАЦИЯ

В последнее время наблюдается быстрый рост числа онлайновых курсов и мест, через которые студенты могут учиться вводный программирование. Как образование программной инженерии становится распространенным в Интернете, онлайн-образование необходимо будет решать, как, чтобы дать студентам навыки и опыт программирования и совместно на реалистичных проектов. В этой статье мы анализируем факторы, влияющие как суперсовместный студия-курс программного обеспечения в кампусе может быть адаптирована как проект-руководимый суперсовместный Mooc.

Категории и Предметные Дескрипторы

K.3.2 [Компьютеры и образование]: Компьютерные и информационные

Наука Образование - компьютерные науки образования

Общие условия Дизайн, Человеческие факторы.

Ключевые слова Разработка программного обеспечения, массивно Открыть Интернет курс, Студия курс, непрерывной интеграции.

1. ВВЕДЕНИЕ

В последнее время наблюдается быстрый рост числа онлайновых MOOCs (широкомасштабных открытых онлайн-курсов) и мест преподавания введения в информатику и программирование. **Coursera, EDX, и Udacity** каждый предлагают несколько таких курсов. В 2012 году, 150,349 студентов, обучающихся в CS50x из Гарвардского университета на EDX, с 1388 получением сертификата об окончании [1]. **Академия Хана, Codecademy, Treehouse** и другие сайты также предлагают студентам поддержку онлайн обучения в программировании. Эти курсы в основном учат студентов программировать самостоятельно. Курс BerkeleyX “Программное обеспечение как Сервис” (Software as a Service) [2] включает в себя совместные темы и призывает парное программирование, но масштаб пройденного студентами сотрудничества мал. Так как образование инженерии программное обеспечение становится все более распространенным в Интернете, мы предполагаем, что онлайн образованию необходимо будет решать, как дать студентам навыки и опыт программирования совместно на реалистичных проектов программного обеспечения. Это было активной темой исследования для курсов в кампусе для более чем двадцати лет [3, 4, 5], но это область онлайн-образования в значительной степени еще предстоит решить.

В предыдущей работе [6, 7] мы описали ход в кампусе сосредоточена вокруг проекта программного обеспечения, который имеет широкое сотрудничество как между, так и внутри группы. Студенты работают в малых группах, чтобы разработать функцию для программы, но должны координировать и интегрировать с многими другими группами также способствует возможности. В 2011 и 2012 году, у нас было около 70 студентов, работающих на одном коде; в 2013 году этот показатель вырос до более чем 140 студентов. Для целей настоящего документа, мы имеем в виду такого рода проекта, в котором студенты сотрудничают в группах, которые, в свою очередь сотрудничают с другими группами, как *суперсовместный* проект.

Мы адаптируем наш курс, чтобы предложить его в качестве Mooc. Онлайн студентам доступны видео, учебники, упражнения, и могут принять участие в суперсовместных проектах. В кампусе студенты будут иметь дополнительные ресурсы времени лабораторной студии и интерактивные семинары и демонстрации преподавателями в классе.

В этой статье мы обсудим соображения, которые влияниют на онлайн жизнеспособность курса-и, как дизайн будет необходимо изменить. До Mooc выполнения мы будем не в состоянии в полной мере оценить результаты, но мы представляем этот документ так, что анализ и вопросы, связанные может быть **переданы сообществу по образованию инженерии программного обеспечения.**

2. РЕЗЮМЕ О КУРСЕ В КАМПУСЕ

Этот курс был разработан в 2011 году, и в течение первых двух итераций было принято преимущественно студентов второго курса программ программной инженерии и информационных технологий. В этой версии курса, студенты сотрудничали в развитии возможностей **для вилки** с открытым исходным кодом проекта Robocode.

Реструктуризация программ школьного уровня вызвала класс, размером более чем вдвое в 2013 году, с добавлением многих студентов разработки мультимедиа и информационных систем. В этой итерации курса, мы изменили проект, чтобы дать студентам гораздо меньшую начальную заказ базу кода для создания из. Проект, который мы выбрали, была построить многопользовательские аркадные игры.

Сотрудничества кода происходит на GitHub, поддерживается непрерывной интеграции серверов, которые строят друг фиксации толкнул на мастер филиала в хранилище. Автоматизированная строит, тестовое покрытие и автоматизированные метрики программного обеспечения дать класса доступ к информации о состоянии здоровья проекта.

Наряду с проектом, еженедельно серию лекций охватывает такие темы распределенного управления версиями, автоматизированная строит, непрерывная интеграция, отладка, тестирование, шаблоны проектирования и другие темы, имеющие отношение к их совместному проекту. Неструктурированной неделю студийная сессия также предоставляется, давая командам недельный тайм-слот, когда они имеют доступ к поддержке в лично в своей работе проекта от воспитателей, преподавателей, и друг с другом.

**В 2013 году 172 студентов, обучающихся в начале семестра, с 146 по-прежнему зарегистрирован в последнюю неделю. За семестр, класс сделал 4883 фиксаций и вырос проект по 67,900 строк кода. Студенты, разработанные включены многие игры, а также сквозные функции, такие как общий наложения, достижений, сетевой поддержки многопользовательской, и запись по событию / воспроизведения**

3. ВЛИЯНИЕ РАЗМЕРА И КЛАССА ИСТОЩЕНИЯ НА ЖИЗНЕСПОСОБНОСТЬ СУПЕРСОВМЕСТНОСТИ

В то время как на кампусе курс 172 можно разместить все студенты на одном проекте, в Mooc, которые потенциально могут иметь порядки больше студентов нужно будет разделить класс на многих когорт проекта.

**MOOCs как правило, имеют также намного более высокую скорость отсева**, чем на кампусе курсов. Приведенные показатели сокращения могут быть 90% или более. Например, из 150,349 студентов регистрации на CS50x только 1,388 (0,9%) получили сертификат об окончании [1]. Для первого курса MITx годов, 6.001x, почти 155,000 регистрирующего только 7157 (4,6%) прошли курс [8], и Coursera Конечно Филипа Зеликов по истории современного мира прошло почти 5000 (10,6%) почти 47000 студентов [9 ].

Большая часть этих потерь происходит рано, и в течение многих MOOCs высокий уровень отсева не может быть проблематичным, поскольку студенты могут получить значение из курса, не закончив его. Для CS50x, только 10905 (7%) студентов представила первый набор проблем, 10137 (7%) указали, что они предназначены, чтобы завершить все работы из курса, и только 3381 (2,3%) указали, что они взяли курс из-за перспективы получить сертификат о прохождении курса [1]. Для 6.001x, 23000 студентов (14,8%) представили первый набор проблем, и 9000 (5,8%) прошли викторину среднесрочный.

Тем не менее, истощение может иметь гораздо более негативное влияние на наш курс. В курсах, которые принимаются в индивидуальном порядке, если студент выпадает они являются только один пострадавших. В команде, конечно, отсева учащихся может привести команде стать нежизнеспособными, оставив своих товарищей по команде "на мель". В суперсовместный если, конечно, слишком много команд по проекту стала нежизнеспособной, проект может стать нежизнеспособными и неэффективными, затрагивая другие команды. Существует вопрос, то, ли высокий уровень отсева из Mooc бы вызвать суперсовместный Конечно, чтобы стать нежизнеспособным. Проекты должны быть управляемыми, когда число студентов находятся на пике, но должны оставаться жизнеспособными и педагогически эффективным, как падают число участников.  
Если жизнеспособность проектов может быть обеспечена, то отсев в командах могут быть смягчены и могут быть использованы в педагогической преимущество. Перераспределение мель членов других команд проверяет как поступающую члена, в понимании новой системы, и команда, в обеспечении прозрачности и документацию продукта и процессов группы. В ходе на кампусе, студенты провести один длинный спринт, позволяя им сталкиваться все содержание курса, прежде чем цикл спринт закончен. Но в онлайн-курс, короткие спринты может быть предпочтительным, чтобы лучше отразить реальную практику и обеспечить естественную точку, в которой перераспределять между командами студентов.  
  
**3.1 Минимальная Педагогически Эффективный размер**

Когда мы разрабатывали курс на кампусе, один из наших принципов проектирования, что студенты должны быть сталкиваются с проблемами, для которых наиболее эффективным решением является использование методов, которые мы учим.  
Это было одной из причин для ввода нескольких групп на одном коде. На небольших проектов, только сотрудничестве внутригруппового, это возможно для студентов, чтобы координировать свою деятельность, просто разговаривали друг с другом, без использования процессов, дисциплина и инструменты, что курс учит. Это когда кто-то вы не говорите на часто начинается модификации кода неожиданными способами, которые вы обнаружите значение хороших тестов, контроль версий, и отслеживания проблем. Требуя межгруппового сотрудничества, мы взобрались на проект далеко за пределы того, что разговор в одиночку могли бы поддержать, и заставил студентов сталкиваются проблемы, что, конечно обучения адреса, а не обойти их. Мы окрестили это "чувствовать боль" педагогика.  
В Mooc, студенты, как правило, удаленных друг от друга, хотя там будут какие группы студентов, которые размещенным. Студенты могут присоединиться к Mooc со всего мира, но некоторые студенты, которые уже знают друг друга, возможно, решит подписать вместе, и местные учебные группы часто формируются.

**Пока проект не состоят только из размещенным команд, было бы трудно для студентов, чтобы координировать через беседы в одиночку. Удаленная связь будет препятствием. Это говорит о том, что в среде Mooc, минимальный размер проекта для того, чтобы быть эффективным педагогически может быть меньше, чем в окружающей среде на кампусе. В трех итераций конечно на кампусе до сих пор, у нас всегда были, по крайней мере девятнадцать команд сотрудничающих на одном коде. Интернет, проект (один коде) потенциально может оставаться эффективным педагогически с гораздо меньше.**

**3.2 Максимальная управляемая Размер**

Если отсеву должно произойти в течение всего курса, это предполагает, что проекты должны будут быть достаточно большой в начале, так что они не дают усадки слишком мал к концу. Тем не менее, даже на пике размера, каждый проект должен быть управляемым.

Есть два аспекта проекта, мы считаем, может стать неуправляемым: код базы, и первоначальный отбор объектов.

**С больше студентов и более команд модификации кода, мы ожидаем, что база кода, чтобы изменить более быстро. Это может сделать это трудно для некоторых студентов идти в ногу с быстро движущейся мишени. Тем не менее, на территории кампуса, мы обнаружили, что код изменяет быстро конце проекта, а команды спешат закончить свою работу до истечения срока.** Рисунок 1 показывает количество фиксаций в неделю в на кампусе курс 2013 году конец из-проекта пик вызывает проблемы для студентов и то, что мы ищем, чтобы уменьшить. Но стоит отметить, что в Mooc это произойдет, когда истощение уже произошло и проекты находятся на их маленький.  
Выбор функций, где группы решили, что они хотели бы работать, происходит раньше в процессе. В то время как группы могут осуществлять более чем одну особенность, как правило, они имеют одну особенность, которая их основное внимание, и, как правило это один они впервые выбран.

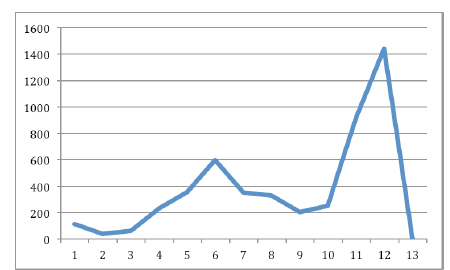


Рисунок 1: Фиксирует неделю в студенческом городке в 2013 году курса.

Этот выбор функция управляет аспекты как сотрудничества и дизайна. Как команды пытаются гарантировать, что их собственная работа является когерентным (поэтому они имеют четкое видение того, что они пытаются добиться) и имеет слабую связь с другими командами (так они не должны ждать от изменения друг друга), команд " Выбор признаков может рассматриваться как быстро установленном дизайн высокого уровня для системы.

В 2013 году, тридцать один команд сотрудничали на проекте. Пятнадцать из этих команд работали на особенностях инфраструктуры и API, которые могут быть потребляемых другими командами, и девятнадцать команд разработаны игры. (Три сделал оба.) В то время как девятнадцать (в 2012 году) и тридцать один (в 2013 году) команд удалось подобрать хорошо облегающую особенности с разумной разделения проблем, что число, вероятно, не наращивать бесконечно. В какой-то размер проекта, было бы слишком много команд в системе для каждого студента, чтобы иметь разумную понимание того, что другие команды выбирают сделать.

На наш взгляд, то, что это выбор, а не функция скорости код, который, скорее всего, ввести верхний предел на размер жизнеспособного проекта.

**3.3 Индивидуальная работа, а также работа в группах**

Одна из стратегий для сведения к минимуму отсев в проектах было бы начать работу проекта после долгих курса истощение произошло. Это также подходит практические потребности структуре проекта.

На рисунке 1, существует сравнительно мало совершает до недели 4. Перед студенты могут начать делать значительные изменения в код, они должны получить его, понять, какую-то часть его, и иметь разумную идею изменения они хотели бы сделать. Таким образом, первая часть, конечно, требует контроля обучения версии (чтобы получить код), создавать системы (чтобы получить код работает), и способы изучения и понимания существующего кода. Это происходит перед сотрудничестве может начать принимать место. **Мы просим, чтобы студенты формируют свои группы по 3 недели курса, и начать установление, что есть они хотели бы работать на неделю в 4.**

**CS50x является Mooc с низкой номинальной ставке завершения, что мы нашли, так делает для подходящего пессимистическому сценарию, чтобы проверить против. В CS50x, 3,292 студентов представил проблему, установленный в 3 недели, и студенты завершили 5,259 последующее викторины [1]. Если мы возьмем, тем выше тех двух чисел (а не регистрации, конечно), как нашу базу, то темпы убыли оттуда на в это 73,6%, а не 99,1%. Это по-прежнему очень высока, но в этом сценарии supercollaboration будет жизнеспособным.**

Цифры 2 и 3 показывают результаты средние двадцати серий компьютерного моделирования, в которой студенты размещаются в командах десяти, с пятьюдесятью команд в проекте. **Моделирование** неоднократно удаляет студента из случайной команде по курсу на отсев. График показывает живучесть команд, и живучесть проектов, где 150000 студентов, помещенных в группах, а затем подвергаются 99% убыли, а где 5250 студентов, помещенных в группах и проходят 74% убыли. В первом случае, половина оставшихся в живых команды остается только один член команды, и половина проектов остается только одна команда. **Тем не менее, в последнем случае, средний размер команда три, и все проекты в каждом счете, по крайней мере четырнадцать активных команд осталось.**

Результаты здесь не предназначены, чтобы быть предсказанием. До сих пор мы очень мало знаем о факторах, которые влияют Mooc цены на отсев, и маловероятно, что студенты будут оставить групп случайно, но можно предположить, что даже при разумно пессимистическому сценарию, supercollaboration не очевидно, обречены к краху. Заблаговременностью индивидуальной работы, прежде чем проект начинается может принести скорость отсева во время проекта ниже скорости, где это жизнеспособно.

**Автоматизированные тесты и упражнения, которые являются типичными для большинства MOOCs, также будет иметь два дополнительных преимущества для курса.**

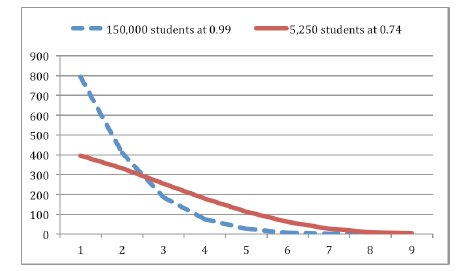


Рисунок 2: Количество команд, по крайней мере п студентов после истощения. Средняя из двадцати компьютерных симуляций.

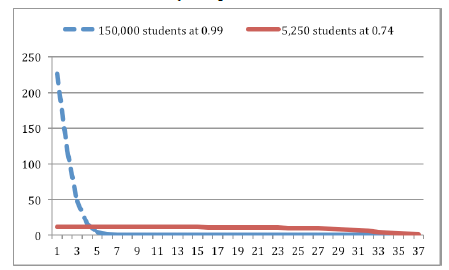


Рисунок 3: Количество проектов, по крайней мере п непусто команд после истощения. Средняя из двадцати компьютерных симуляций.

Во-первых, подвергая деятельности учащихся (хотя и не их знаки) в отдельных упражнениях, курс может стимулировать активных студентов в кластер вместе в командах. Это обеспечит дополнительную защиту от отсева, а вред возникает, когда активным студентом считает, что его или ее сотрудники являются неактивными.

Во-вторых, это позволит улучшить поддержку Курса для опробовать технику в ручной среде, перед применением ее к проекту. Проект заставляет студентов применять некоторые понятия часто, как слияния изменений, и производит немедленно видимые результаты для них. Тем не менее, на кампусе, заметим, что другие темы (например, рефакторинга) имеют более длинные петли обратной связи. Команды, потребляющие изменение кода, возможно, не будет готов к работе с ним немедленно, и студенты не всегда просматривать отчеты статического анализа. Автоматизированные упражнения обеспечит студенты могут проверить свои знания по этим темам с немедленной обратной связи.

**4. избирательным участием и неоднородность**

В традиционной на кампусе конечно, знания, фон, и интересы студентов в значительной степени ограничены. Мы знаем, что студенты взяли предварительные предметы, и что они не приняли последующие предметы, для которых это является необходимым условием. Мы знаем, что, как студентов, они будут иметь широкий фокус на большей части тем, мы требуем к экзамену, потому что они должны пройти их курсы, чтобы получить их степеней. В Mooc, это не тот случай. Некоторые студенты, возможно, уже градусов, в то время как другие все еще может быть в школе.

**В 2013 году наша кампус на группы стал несколько более разнообразны, как мультимедийный дизайн и информационные системы студенты пришли в ходе наряду инженеров программного обеспечения. Для удовлетворения этого, мы изменили проект от улучшения уже существующие продукт (RoboCode) для создания новой системы с очень небольшой базы. В качестве основания была небольшой, это было сравнительно непринужденно, и команды смогли специализироваться. Некоторые команды взял его на себя, чтобы определить язык дизайна аркады, что команды должны вписаться в. Другие команды построена игры. Другие Встроенный инфраструктуру, которая используется исключительно за кулисами. С точки зрения баланса команд, мы рассматриваем это как успешного изменения.**

Для Mooc когорты, которая еще более разнообразны, курс будет нужно быть более гибким по-прежнему. Некоторые студенты могут быть только заинтересованы в конкретных тем в рамках курса, и не имеют времени для проекта. Другие могут быть хорошо разбираются в материале, но, возможно, заинтересованы в участии в проекте со студентами для того, чтобы попытаться взять на более старший роли в команде, чем они есть на своем рабочем месте. В нашем дизайне, суперсовместный проект будет крупнейшим компонентом курса. Обучение будет включать в себя содержание тема на основе последовательности видео и упражнений, с каждой теме последовательности в состоянии стоять в одиночку, но также являющиеся частью последовательной повествования в курсе. Цель в том, чтобы гарантировать, что новые последовательности тема может быть добавлен по мере необходимости, без Конечно становится не пересекаются.

В ходе разработки программного обеспечения, как это, есть и другие, чем персонал, конечно, что бы представлять интерес для студентов много голосов. Есть много профессиональных материалов, видео, презентации, и все большее количество онлайн игровые площадки и учебные пособия доступны. В предыдущей работе, мы предложили курсы, где умные учения, поддерживаемые динамическую экосистему социально обнаружили, написано, и добавил, содержание [9]. Цель динамично развивающихся курсов еще далеко, но мы можем начать с в том числе интервью с практикующими из компании и проекты с открытым исходным кодом, и других материалов от людей за пределами курса.

5. **КОЛЛЕГИАЛЬНАЯ** МАРКИРОВКА ГРУПП

В 2013, конечно, мы ввели **критику (рецензирование)** в неделю 8. Группы представили свои успехи, и искал обратную связь от своих коллег. Студенты были выделены пять групп, чтобы отметить и **критику,** заполнив простое обследование. Текстовая обратной связи в каждой критике затем доступны к группе критики, которые, в свою очередь отметит критику на, было ли это цель, конкретные, действенные, конструктивной, и полезно. В общей сложности 725 критик были завершены, и 3,075 отзывов о критиках.

Только вмешательство сотрудников мы были призваны осуществлять на маркировку критики (кроме написания программного обеспечения для его поддержки) был обработки небольшое количество запросов, чтобы исправить ошибки студента - особенно там, где студенты представили критику за одну команду, что они предназначены, чтобы быть для другого. Тем не менее, поскольку каждая группа получила более двадцати критик, даже если не принять эти ошибки должны были бы незначительное влияние на маркировке.

**В нашем кампусе на процессе, конечные результаты проекта команд отмечены персонала курса. В командах, студенты используют систему коллегиального оценивания разделить свой след проекта между ними в соответствии с их вкладом. Однако, похоже, что это будет жизнеспособным заменить персонал маркировку с последовательностью коллегиальной критики, которые также могли бы стимулировать студентов, чтобы не оставить так много работы для лихорадки окончательного недели.**

6. ЗАКЛЮЧЕНИЕ

Supercollaboration в классе является сравнительно новым. Насколько нам известно, наш курс по-прежнему является единственным программным обеспечением инженерной Конечно, чтобы поместить так много команд студентов на одном проекте. Supercollaboration в открытом классе онлайн, то, всегда скорее всего, быть на шаг в неизвестность. Тем не менее, наш анализ показывает, что это возможно, и мы находимся в процессе получения такого курса. Ключевые аспекты, кажется, сохраняя элементы курса когерентного но независимы, и обеспечить, чтобы проектные группы только форму после начального высокого студента истощение произошло, когда есть данные для идентификации и группа активных студентов.

7. Авторы

Мы с благодарностью признать помощь наших курсов воспитателей, Джексон Gatenby и Филлип Дрю. а также Йорн Ги Су., с которыми мы сотрудничаем предназначен первой итерации курса. NICTA финансируется правительством Австралии в лице Департамента Broadband, связи и цифровой экономики и научно-исследовательского совета австралийского через ИКТ центра программы совершенства.