A Case Study in Teaching Agile Software Product Line Development

Dr. Derek David Riley, Milwaukee School of Engineering

Dr. Riley completed his PhD work in modeling and simulation at Vanderbilt University in 2009 and has expanded his scholarly and professional activity to include mobile computing and software engineering. He is currently a faculty member at the Milwaukee School of Engineering.

A Case Study in Teaching Agile Software Product Line Development

Derek Riley
Milwaukee School of Engineering
riley@msoe.edu

Abstract
The development of Software Product Lines (SPL) hold promise to improve the efficiency of writing and maintaining large software projects, but SPL engineering can be difficult to teach in a software engineering classroom for many reasons. The development of a non-trivial SPL typically takes longer than the time available in a typical semester, student interest in SPL engineering is rarely inherent, and learning outcomes from different approaches to SPL engineering are not always consistent or aligned with traditional software engineering learning goals. Further, applying SPL methods in an agile development environment can be challenging because agile methods typically prioritize features and bug fixes over SPL (maintainability) goals.

In this work we investigate a couple of research questions including: can incorporating SPL into an undergraduate software engineering course sequence improve student learning outcomes related to writing maintainable, reliable, and reusable code? We are also interested in answering the question of whether incorporating SPL can improve the quality of documentation created by students. To work toward answers to these questions we present a case study of the two-semester development of an SPL in a senior-level undergraduate software engineering course sequence using a real-world mobile application. We present key strategies for motivating positive learning outcomes including an adapted Scrum process designed to incorporate SPL engineering. We found that our approach improved student application of reusability theory, benefitted documentation quality, increased student satisfaction with the course, and increased the percentage of code written reused by students from semester to semester.

1. Introduction

A Software Product Line (SPL) is a set of software systems that share common assets and are easy to deploy and configure for new environments [6]. There are many approaches that can be used to create an SPL including model-driven development, modularization refactoring, the use of SPL design patterns, reuse design, and others, but few software engineering classes have time to teach or apply these concepts. Many of the strategies within SPL engineering reinforce good software engineering practices such as reusability, maintainability, and testability, so incorporating these concepts using an applied project in a software engineering course is beneficial to students to prepare them for programming challenges they will face in the future [18]. An inability to effectively apply SPL engineering concepts can lead software engineers to build software in a way that leads to inefficiencies, time overruns, and difficult-to-maintain software [14,18].

Development of an SPL for mobile applications is attractive due to the need to support multiple (Android, iOS, and potentially others), regularly-updating operating systems to reach the largest number of potential users. Differences between screen sizes, orientations, and features make developing and maintaining an SPL for mobile platforms especially challenging. Fortunately, the challenges are easily understood by students, and the benefits can also be realized in a class. Hands-on teaching of the basic strategies of SPL engineering, seeing good mobile examples, and using standardized libraries instead of writing custom code are critical to making mobile SPL engineering work in the classroom [4].

Scrum is a commonly-used software development process that lends itself well to mobile development [20], and students typically prefer using an agile methodology to traditional processes [15]. Unfortunately, the product owner in Scrum is left to make decisions about priority of tasks for the team, and unless the product owner is SPL savvy, new features are easily prioritized over SPL engineering. Existing SPL development strategies exist, but applying the approaches in a

classroom environment can lead to difficult-to-maintain software and limited learning outcomes [18].

Creating a cohesive, productive team is a known challenge in software engineering classrooms, and while many strategies exist to improve productivity and grades for software engineering teams, this problem continues to persist. It is also easy for weak team members to hide behind stronger programmers. Many strategies exist to address this challenge, but it still persists [11].

In this work we investigate a couple of research questions including: can incorporating SPL into an undergraduate software engineering course sequence improve student learning outcomes related to writing maintainable, reliable, and reusable code? We are also interested in answering the question of whether incorporating SPL can improve the quality of documentation created by students. To work toward answers to these questions we present a case study of four years of teaching a two-semester course at a primarily undergraduate institution where students started and continued development of an SPL using a real-world project.

The class worked as a collaborative team on mobile applications (Android and iOS) as well as a web database that helps local transit riders find when the next bus is arriving at their location. The two-semester course sequence was originally designed to teach applied software engineering, and over the four years of this case study, an adapted Scrum process as well as SPL engineering were incorporated to improve student learning outcomes. The original transit app software developed by the first class (using Scrum without any SPL focus) took over two months to re-deploy for a new transit system. Updates made by subsequent classes where SPL practices were emphasized have improved the re-deployment time so that current versions of the software can now be redeployed in a matter of minutes.

In the process of re-deploying the transit app, we developed an adaptation of Scrum to incorporate the development of SPL practices into the traditional Scrum roles and documentation. Our adaptation of Scrum adds an SPL owner (usually the instructor), who balances the needs of maintainability goals with new features and bug fixes for the team, as well as some additional documents and ceremonies.

We present key lessons learned through this process of teaching SPL engineering using Scrum for an applied project. We found that our approach improved student application of reusability concepts, documentation quality, student satisfaction with the course, and increased the likelihood that code written by one semester's class will be reused by students in a future semester.

2. Related Works

Planning for SPL development has been analyzed from several viewpoints including: asset developer [10], requirements engineer, and product developer [8]. The application of method engineering approaches to SPLs have allowed SPLs to encompass a more comprehensive view of the software to improve consistency and alignment with goals for the software [8], and the application of SPLs has been done in the classroom in dedicated classes. Effective planning for SPL development requires familiarity with relevant SPL methods and the core product being developed [18].

SPLs have been developed for mobile applications previously. A product line architecture was developed for a role-playing game on an early smartphone that allowed developers to improve performance and development speed through the incorporation of SPL development methods [21].

Combining incremental prototyping and plan-driven development processes has also proven effective for an SPL in a mobile game environment using aspect-oriented programming [1].

Agile development processes such as Scrum [20] are growing in popularity, and their combination with reuse-oriented development has been studied previously. Encouraging reuse can be challenging in agile methods [5], but strategies that combine long-term reuse strategies into agile methods using feature-orientation has been shown to improve product reuse without compromising agility of the process [12]. SPL methods have been previously combined with agile processes such as Scrum with positive results despite the inherent clash of priorities in SPL techniques and agile methods [16]. However, these modifications to Scrum can add significant overhead and limit the viability of the approach especially in a classroom. Further, it can be difficult to prioritize SPL engineering over new feature incorporation when task prioritization is driven only by the product owner. While agile development methods and SPL engineering share similar goals, incompatibilities exist, due to the long-viewed nature of SPL engineering and the lightweight and short-term planning required of many agile development methods [17, 16]. However, using feature-driven development, SPL have been implemented and have been shown to be compatible with certain agile methods in industry, but a full team understanding of SPL is required to achieve the benefits of the approach [17].

Software reuse is an important topic in SPLs, and several papers have addressed software reuse from various perspectives. The concept of reusing more than just code is critical [21,13]. Agile development does not always encourage reuse, and the classroom environment inherently deemphasizes reuse of software due to guaranteed turnover of the teams, so care must be taken to encourage reuse engineering when developing software using these methods.

Mobile application development is a relatively young field, but some research has been done in how to teach students about mobile application development to prepare them for challenges they will face or how to learn a new programming language [4]. One promising approach incorporates "Challenge Based Learning" into Scrum to better prepare learners for the obstacles they will face in real teams [19]. A drawback of this approach is that unless reuse coordination is emphasized throughout the semester, it is easy for young developers to ignore reuse in favor of completing other features or bug fixes.

3. SPL course

Software engineering is a commonly-required course or pair of courses in many computer science programs, and learning goals for these courses tend to be similar. At UW-Parkside, software engineering is taught as a senior-level course sequence over 2 semesters and students are expected to work on a real-world project to gain practical experience that will prepare them for graduate school or future careers.

The UW-Parkside the courses aim to educate and provide experience for students working as a team, using various software process models, doing requirements gathering, exercising design concepts, applying testing approaches, and understanding maintainability concepts. Many software engineering courses have one or multiple community-based projects that have non-technical clients the students have to interact with to gain experience with requirements elicitation, testing, and other communication and UW-Parkside uses this strategy to further develop students' experience.

Incorporating an SPL focus into the class was an accidental strategy developed in a response to poor learning outcomes in testing, maintainability, and documentation creation. Prior to the SPL

focus, students could answer basic exam questions on testing, reusability, and maintenance, but they struggled to apply the concepts to projects. We found that students were eager to learn what they needed to achieve the grade they wanted on homework, quizzes, and tests, but they rarely were able to effectively apply the concepts to their projects.

In response to the realization that students had these poor learning outcomes, we chose to try continuing a project from semester to semester and focus on code reuse and maintainability. To formalize the efforts, we attempted to apply formal SPL concepts in the classroom within the Scrum framework and compare outcomes with and without the SPL efforts. Only one section of software engineering courses is taught each year at UW-Parkside, so the comparisons were made between previous years (without SPL) and more recent offerings (with SPL).

Students are not explicitly told that the class is developing an SPL; instead they are given code and documentation from previous classes at the beginning of the semester, and they have to learn how to deploy it and use it from day one. Students quickly understand the need for documentation and as they start working with the software, and they understand the importance of writing reusable code as they are forced to put in extra effort to rewrite parts of the code that aren't consistent with the documentation. Additionally, we establish metrics for SPL quality (such as amount of time to deploy) from the beginning to establish a clear metric for success or failure. To continue to emphasize the importance of SPL goals during the semester, we have made modifications to the Scrum methodology, which we describe in a later section of this paper.

As part of the SPL development approach, we use modern coordination and collaboration tools including Slack (slack.com), Trello (trello.com), and Github (github.com), which are free to use and facilitate management of communication between team members, documentation, code management, and bug tracking. We have found that these tools are essential to the learning outcome improvements because they are more relatable to students who are increasingly expecting instant feedback communication and the ability to work at any time from any location. Further, these and similar tools are used in industry, so it is beneficial for students to gain experience with them.

To assign grades in this course, we use 3 components: quizzes, project work, and a final exam. Weekly quizzes keep students aligned with course lecture topics. The project work is assessed through establishment of expectations for a certain amount of effort (hours and LOC) put forth toward the project. Students are expected to put effort toward every phase (requirements, design, implementation, testing, and maintenance), and they are required to document their effort in a digital journal. Students also are required to write a final reflection that describes their contributions to the project and learning outcomes. The journal and reflection information is required to be aligned with information posted on Slack, Trello, and the code repository.

4. Project Description

Mobile applications are an excellent project for students to learn software engineering and SPL because of their complexity and the natural inclination of students to be interested in mobile applications. Further, online resources are good enough that a lack of initial experience in mobile development for instructors and students is not a barrier to success in the project.

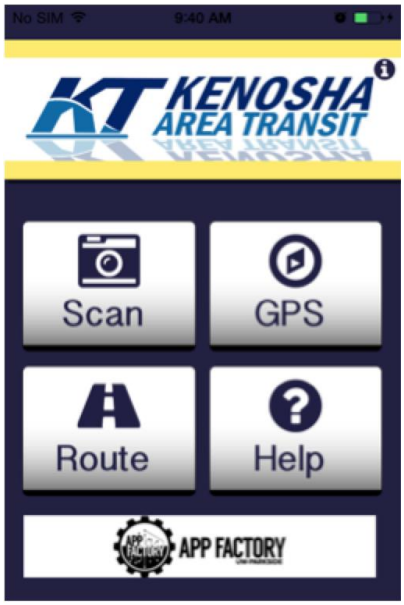

Figure 1. Screenshot of the iOS version of the TA home screen for the local transit system.

4.1 Domain Description

Fixed-route bus systems exist in most urban areas in more than 1,200 fixed-route bus systems in the United States ranging from hundreds of bus routes to only one route [2]. Prior to the introduction of mobile technology, fixed-route bus systems relied upon paper schedules in brochure or poster format to communicate route information to riders. Today, most bus systems (approximately 85%) maintain their own websites where schedules are posted. However, the complexity of the schedule combined with small screens of mobile devices make it difficult for most people to easily find the information they want when they are at a bus stop. Many large bus systems have implemented mobile apps or Google Transit to help riders determine when the next bus is arriving at a given stop [7], but smaller transit systems often cannot afford the cost or staffing required for these solutions.

The fact that many students are familiar with buses and transit systems makes this an appealing project because it is easy to describe the problem and understand the motivation for the development goals. Also, the need to support multiple platforms (iOS, Android, web) is a realistic challenge that exposes students to tradeoff decisions that they will face in their future careers as software engineers. Close interaction with the transit system helped the students see how the software is used and helped us better define the SPL assets as they were developed.

4.2 Product Architecture

The product we developed for the transit system consists of several components and is designed to be easy to update for the transit system. Riders of the busses can use the Transit Application (TA) with or without an internet connection. The TA is deployed natively for Android and iOS and includes database on the mobile device for offline functionality. A screenshot of the main page of the app for the original deployment is shown in Figure 1.

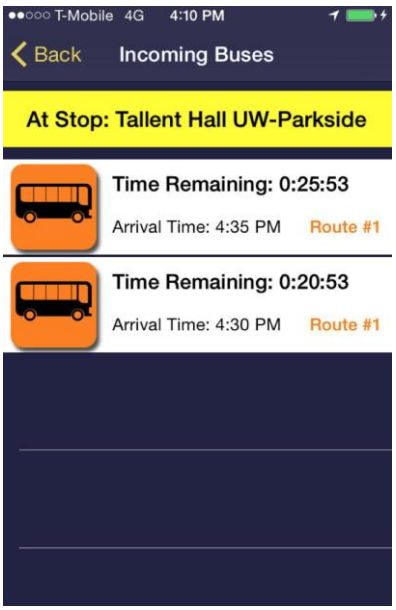

Figure 2. Screenshot of the iOS version of the TA stop screen that shows the arrival times of the busses.

We have attempted to employ the "Product Parts" SPL pattern to develop the core assets of our application [9]. The TA application includes three main functional features and a help page as shown in Figure 1. The "Scan" feature allows users to scan a unique QR code at a bus stop to find out when the next bus is arriving on any of the bus routes that serve the specific stop. Information on the results page is provided to the user as a sorted list of stop times (see Figure 2). The "GPS" feature allows a rider to take the current GPS location of their phone (using the GPS localization built into the device) and determine the closest stops and when the busses are arriving at those stops. The "Route" feature utilizes Google Transit to allow a user to find a route from their current location to a destination location. This tool allows a rider to determine how to transfer between busses, routes, and other modes of transportation to reach their destination.

The TA Manager is a web-based tool designed for the transit administrators to keep the routes and schedules updated on the mobile apps. The TA Manager pushes updates to the TA apps whenever changes are made to the database to ensure the most accurate data is available. The TA manager tool keeps an up-to-date version of a database with all the routes and it also allows the transit managers to import and export the standardized General Transit Feed Specification (GTFS) files, which are needed to keep Google Transit up-to-date [7]. The TA Manager can also host public mobile-friendly websites that provide similar information for riders who use Windows, Blackberry, or other mobile platforms. A diagram of the organization of the product structure is shown in Figure 3.

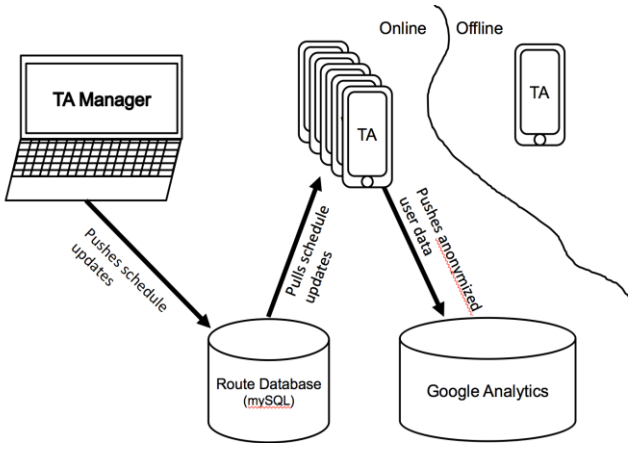

Figure 3. Overall system architecture.

TA apps, while disjoint by the different languages for the different platforms (Java and Swift), are joined under similar branches of logic. Hosted on each application is an instance of a database, which mimics the GTFS file structure and uses platform-independent queries.

4.3 SPL Assets

Within the TA Apps, we have identified several components as SPL assets. The four buttons seen in Figure 1 identify four assets that can be included or excluded as requested by a transit system. Each of these features (in Android and iOS) is developed to be modular so that they can be easily modified or eliminated for future deployments. Our SPL approach was adapted with the mindset of utilizing a Model View Controller (MVC) architectural pattern as closely as possible.

Our approach allows for rapid deployment of applications because only small parts (less than 0.01% of the total LOC) of the application must be rewritten when individual changes happen to the operating system, user interface, modules, database, or underlying logic. The TA Apps currently have approximately 30,000 LOC, of which approximately 20 need to be changed for a redeployment.

The TA Manager has two central objectives: to allow the transit authority's authorized users the ability to load transit data onto the system and to translate the data into forms for machine and human consumption. All data is stored in a database, in which the schema is based on the GTFS with additional enhancements for specific use-cases that enhance usability for transit system managers.

The size of the TA Manager is more than 100,000 LOC (some of which are auto-generated or imported from libraries). To adapt the manager for another system, less than ten LOC need changing in addition to the database (which is updated automatically using the GTFS files) and the logos branding the application.

There are many potential future product feature variations that could be included in our SPL for future sections of the class. We are currently developing a crowdsourced, real-time positioning tool for the application using Bluetooth beacons that will allow real-time GPS locations of the busses without using an existing AVL GPS system. The real-time data can be added as a module to the pre-existing GTFS real-time specification to maintain standardization and easy integration for the SPL.

Another potential feature that can be included as an SPL asset is advertising. Transit systems often rely on advertisers to offset costs, so they have existing advertising frameworks, and ad space on the mobile apps could provide a valuable revenue stream to offset costs of maintaining the application. The last two future features that are natural SPL assets that could be developed for transit systems are mobile bus passes and mobile payment systems.

5. SPL agile approach

Agile methods are an efficient development process for developing dynamic software products, and their use has been gaining popularity, especially in the classroom [15]. Agile methods encourage emphasis on establishing requirements for a single customer (the product owner), but SPL development requires an understanding of which assets are useful for other potential customers, which creates an inherent challenge when developing an SPL in a classroom due to the lack of domain knowledge [16].

5.1 Scrum SPL Asset Identification

Identifying the core features of the application was relatively straightforward as the product owner clearly identified them through the traditional Scrum user stories. However, to identify the SPL core assets, multiple client perspectives is needed. To accomplish this in the classroom environment, we commissioned a market study (through a collaborating marketing research class) and identified common features in existing mobile applications to determine the core SPL assets.

We attempted to accomplish these SPL asset discovery tasks within the traditional Scrum framework, but found that it was easy for the SPL components to be de-prioritized compared to the needs of the original product owner. We completed the SPL core asset analysis simultaneously with the Scrum process, and we modified and added to the user stories in the product backlog identified by our product owner. Our modifications were performed during the "after lunch" portion of the planning meeting (when the product owner was not involved) to ensure the SPL modifications were included as user stories and not under-emphasized by the product owner. While the modification and addition of the user stories adds overhead to the development process and extra work for the developers, incorporating the changes as part of the Scrum sprints provides an opportunity to talk about the bigger picture of the development process, which is important for students to discuss.

5.2 SPL Scrum Variation

Through our experience we propose that the traditional Scrum sprint planning meeting can be augmented with an additional task: SPL planning. By incorporating SPL planning as well as a dedicated team member responsible for the SPL, the SPL owner, Scrum can naturally incorporate the development of an SPL simultaneously with an agile product without adding significant overhead to the Scrum process.

The SPL owner should have a variety of skill sets and responsibilities, and the SPL owner role can be added to Scrum master duties or a separate team member could be designated as SPL owner. This role is best held in the classroom by the professor due to the experience required. The SPL owner identifies SPL tasks during the planning meeting (the after lunch portion) and makes sure that the tasks are added to the product backlog and appropriately ranked. The SPL owner also aids in planning for the sprint backlog and separating tasks into manageable sizes. The SPL owner is in charge of training and motivating other team members to ensure that the SPL tasks can be completed.

Traditional Scrum product backlog refinement is accomplished during a product backlog planning session involving the product owner, Scrum master and development team. We propose that the SPL owner also be involved in the planning process to identify SPL components to include in the sprint backlog.

In Figure 4, we show our modified Scrum sprint structure including the backlog artifacts. It can be seen that SPL user stories are identified during a project planning meeting and added to the product backlog and sprint backlog. These stories are not prioritized higher than the development tasks but are still included in the sprint. Clear identification of the SPL tasks and artifacts allows for better separation of the core assets from the specialized product. Inherently, some SPL tasks are comingled with development tasks (for example, following a design pattern such as MVC), so the SPL tasks that are part of development tasks are incorporated accordingly into existing user stories as part of the sprint planning meeting.

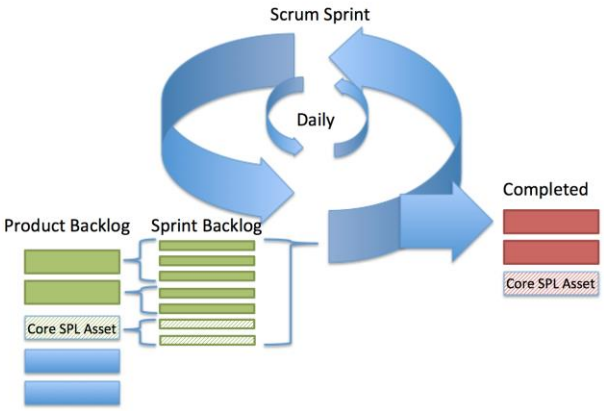

Figure 4: Scrum sprint modification to include core SPL asset development.

5.3 SPL Deployment

The original development took one semester (four calendar months) with a class of 20 students to complete using Scrum without a focus on SPL engineering or maintainability. Soon after the first deployment, it was clear that the software would need to be re-written to fix some fundamental structural bugs and incorporate standards if it were going to be useful for other transit systems. The software was re-written and redeployed for the second transit system over the course of 2 months in an independent study with a team of five students. During this re-write, we began to employ and refine the incorporation of the SPL owner into the Scrum process. The first redeployment of the apps and manager after the SPL-focused rewrite resulted in a deployment time that took less than one day. Subsequent SPL development in future software engineering classes has reduced the deployment time to under 15 minutes. The progress of the deployment improvement is shown in Figure 5. It is clear from this improvement that students were able to not only write more reusable code, but while doing this the quality of the documentation improved and the students better understood real-world applications of the SPL concepts.

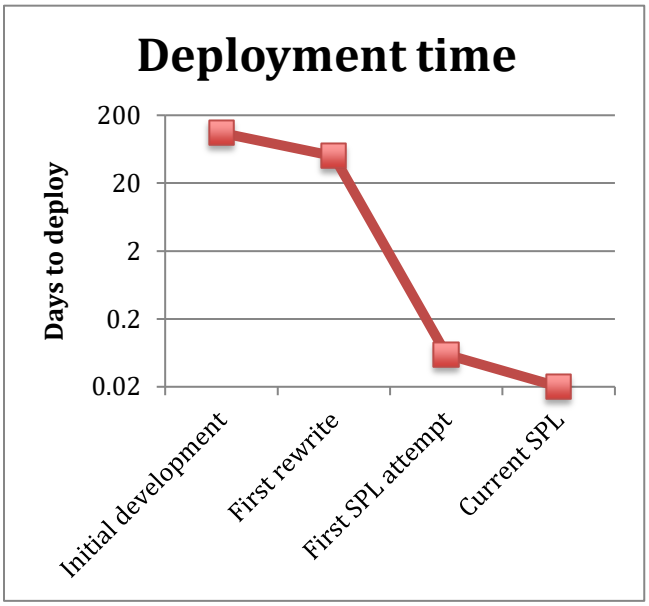

Figure 5. Chart of deployment time in days for the apps and manager for the phases of the project.

6. Lessons learned

In the process of developing this SPL, we learned many things about designing, deploying, and maintaining an SPL using Scrum in a classroom environment. We also learned many things about our original research questions.

6.1 The motivation must be clear

A motivation for the project needs to be established for the team early in any project, and while users of a product can demonstrate their need, demonstrating the need for SPL engineering doesn't usually come from users. Theoretical arguments about software maintainability are difficult to grasp for students, especially when much of their experience in other classes is developing software to be turned in and discarded after grading.

We found that attempting a timed deployment of the previous software early in the semester and discussing the shortcomings is a great motivational tool for students to understand why incorporating SPL user stories is critical. Students are typically eager to be critical of previous classes' code and approach, so it is easy to facilitate this experience to start the semester. Students often will cite the need for more documentation, which motivates them to create such documentation and understand its importance from the beginning. Balancing the SPL and new feature motivations is a challenge for the SPL owner, but this can be adjusted throughout the semester as the project evolves. Regular deployment tests coupled with user tests keep the motivation fresh throughout the semester.

6.2 Ad-hoc agile methods in the classroom are not likely to lead to re-deployable systems

The initial prototype of a new piece of software is inherently focused on developing features, so the agile approach of "developing for the demo" can easily lead to code that demonstrates functionality but is not maintainable. Experienced developers who have a focus on developing maintainable code can avoid this trap, but new software developers often struggle to balance SPL development and feature development. Inherently, feature development is prioritized over SPL assets in Scrum because SPL assets are difficult to "demo" in a traditional sense and are de-prioritized if they are acknowledged at all.

The traditional response to this challenge of developing demo-able but not maintainable code is to spend significant effort refactoring a project, or throw out the original project and start from the beginning. Our approach of incorporating an SPL owner into the project changes the motivation discussions and leads to more software reuse. We found that mindfulness of SPL and maintainability by the whole team from the beginning of a project does delay development of features, but ultimately it saved time not having to re-write or refactor as often. Students initially were skeptical of the additional costs of SPL engineering, but as they saw the deployment tests, they appreciated the importance of maintainability more, and many cited it as the most significant learning outcome of the course.

6.3 Competition is a strong motivator

By continuing a similar project from semester to semester, students were not only exposed to the good and bad coding of previous teams, but they were also given an opportunity to compete with their peers. Trying to add features and decrease the deployment time to beat their peers was an incredibly strong motivator for many students. Having a simple metric such as deployment time makes the success criteria easy to measure and compare, which further enhances competition.

We observed increased creativity and ownership of the project by using competition as a motivator, which further improved the learning outcomes and experience of the team. While not all students are motivated by competition, the agile approach of collective ownership facilitated the competitive students' motivation of the non-competitive students.

6.4 Effective tool use is critical

Using tools like Slack, Trello, and GitHub not only mimic what students will be seeing in their future career, they also reinforce dedication and flexibility of the project. The ability for students to collaborate remotely and understand the challenges of doing so, especially when the project is focused on reusability is another major learning outcome of the course. Keeping a watchful eye on the communication and repository use is an important tool for the instructor to not only identify students who are struggling, but it also allows an additional mechanism for providing feedback to students and motivation to contribute fairly to the project.

Students universally appreciated the importance of the tools by the end of the class (as indicated in a post-course survey), and they demonstrated a mastery of the tools within a few weeks of starting the course. The tools clearly supported the learning outcomes for the class, and also helped track metrics supporting accountability.

6.5 Consistency is key for learning maintainability and SPL engineering

The initial deployment of our project relied on individual developers to make their own decisions regarding the maintenance of the project. This resulted in a fragmented code base that contained several pieces of the project that were individually maintainable but when combined led to an integration that was undeployable or unmaintainable for future teams. The inconsistency of

individual approaches can be mitigated through code standardization and experience, but young developers need careful guidance to develop these skills, and it is difficult to achieve these goals in a single course.

Consistency and standardization can be established and led by the SPL owner and teams adopt the philosophies started by good examples. We realized an additional benefit emerges when developers are actively guided by an experienced SPL owner (who continues with the project from year to year): the likelihood that the SPL assets will be used in the future is increased. Since developers are more invested in and familiar with the SPL features, it becomes much easier to encourage their use in the future (even semester to semester), which can be a major challenge for traditional SPL approaches. Increasing the likelihood of reuse is possibly the most important learning outcome of our approach (and therefore the most important answer to our research questions). While it is natural that a design will stabilize over time, the SPL focus ensures that an emphasis is placed on reuse, which motivates better design earlier.

6.6 Student learning outcomes do improve using SPL in the classroom

The approach we used to introduce students to SPL concepts resulted in not only more efficiently-deployable code, but we also saw a dramatic improvement in the quality of documentation created. While quantitative data on this trend is difficult to gather or draw conclusions from, there are many qualitative observations that demonstrate the validity of this approach to improve learning outcomes.

It was observed that students in general took a longer view of the lifespan for the software and were more mindful to create better comments, clearer documentation, and many were eager to continue to work on parts of the project even after the course was over once the SPL concepts were emphasized. Comment percentages increased from the single digits to double digits, and more students utilized the comments that were written (observed informally). Some of the documentation and efficiency improvements can be credited to general maturing of the software, but generally the focus of the discussions students had in planning and review meetings changed to focus more on long-term goals after the SPL focus was added to the class.

The authors also noticed a clear differentiation in the learning outcomes and ability to apply theoretical concepts to the project once the SPL concepts were emphasized in the course. Students not only answered the basic questions about maintainability and testing, but they were able to apply the concepts more concretely in open-ended questions on quizzes and exams. This led to a noticeable improvement in grades on these specific assessments over the span of the case study.

7. Conclusions

In this work we presented the development and re-deployment over multiple years of an SPL for transit systems software in a classroom environment. We also presented our adapted Scrum model that incorporates changes to the Scrum model to incorporate SPL asset recognition and development through the SPL Owner. Originally we aimed to understand whether incorporating SPL into the software engineering courses would improve student learning outcomes related to maintainability, reliability and reusability. We were also hoping to determine whether the curricular changes could improve documentation quality and code redeployability.

Ultimately, we found that by following our adapted SPL Scrum approach, subsequent semesters of students continually improved deployment times by multiple orders of magnitude and increased code reuse incrementally. Student learning outcomes from course assessment averages (quizzes,

exams) improved, and the standard deviation on in-course assessments decreased, indicating that more students had a more consistent understanding of the concepts.

One area that we felt could have been better addressed was software testing. With the additions of the SPL focus, less emphasis was put on formal testing methods, which showed in early sprints. As with Scrum, our process was adapted over time using the retrospective mechanism, which helped us address the shortcomings of the approach and tailor it to the specific group of students. In future offerings, a stronger emphasis to generate automated tests within a formal testing framework would likely further improve the code quality and improve student productivity.

The vast majority of students cite their project experience in this class as the most important they had while in college, and the favorability ratings for the project increased in post-course assessments after the introduction of the SPL. While the example we present in this work has fortuitous SPL properties that allowed for straightforward identification and deployment of SPL assets, we feel that our approach could be applied to other applications with positive results, especially in the classroom environment.

8. References

[1] V. Alves, P. Matos, L. Cole, P. Borba, G. Ramalho, *Extracting and Evolving Mobile Games Product Lines* LNCS, Vol 3714, pp 70-81, 2005.

[2] American Public Transit Association 2014. *2014 Public Transportation Fact Book*. Washington D.C.

[3] K. Beck, et al, *Manifesto for Agile Software Development*, http://agilemanifesto.org, 2001.

[4] D. Riley, *Using mobile phone programming to teach Java and advanced programming to computer scientists.* Proceedings of the 43rd annual SIGCSE, 2012.

[5] R. Carbon, M. Lindvall, D. Muthig, P. Costa, Integrating product line engineering and agile methods: Flexible design up front vs. incremental methods, *First International Workshop on Product Line Engineering*, 2006.

[6] Carnegie Mellon Software Engineering Institute, URL: https://www.sei.cmu.edu/productlines/

[7] M. Catala, S. Dowling, D. Hayward, *Expanding the Google Transit Feed Specifications to Support Operations and* Planning, No. FDOT BDK85# 977-15, 2011.

[8] G. Chastek, P. Donohoe, J. McGregor, and D. Muthig, Engineering a Production Method for a Software Product Line, *SPLC*, 277-286, 2011.

[9] P. Clements, L. Jones, J. McGregor, L. Northrop, *Getting There from Here: a roadmap for software product line adoption,* Comm of the ACM, (49) 12, 33-36, 2006.

[10] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*, SEI Ser. In Soft. Eng. Addison Wesley, 2001.

[11] M Marques, *Monitoring: An Intervention to Improve Team Results in Software Engineering Education*, Proceedings of the 47th SIGCSE, 2016.

[12] M. Kircher, P. Hofman, Combining Systematic Reuse with Agile Development- Experience Report, *SPLC*, 2012.

[13] A. Martini L. Pareto, J. Bosch, Enablers and Inhibitors for Speed with Reuse, *SPLC*, 116-125, 2012.

[14] T. Mende, R. Koschke, and F. Beckwermert, An Evaluation of Code Similarity Identification for the Grow-and-Prune Model, *Journal of Software Maintenance and Evolution: Research and Practice*, 21(2):143-169, 2009.

[15] L. Mikael, D. Muthig, A. Dagnino, C. Wallin, M. Stupperich, D. Kiefer, J. May, T. Kahkonen, *Agile Software Dev in Large Organizations*, Computing Practices, 37(12), 38-46, 2004.

[16] P. O'Leary,F. McCaffery, S. Thiel, and I. Richardson, An agile process model for product derivation in software product line engineering. J. Softw. Evol. and Proc., 24: 561–571, 2012.

[17] R. Paige, et al. "Towards an agile process for building software product lines." *Extreme Programming and Agile Processes in Software Engineering*. Springer Berlin Heidelberg, 198-199, 2006.

[18] J Hunt, J McGregor, Software Product Lines: A Pedagogical Application, *JCSC,* 22(2), 2006.

[19] A. Santos, A. Sales, P. Fernandez, M. Nichols, Combining Challenge-Based Learning and Scrum Framework for Mobile Application Development. ITiCSE, 189-194, 2015.

[20] K. Schwaber, M. Beedle, Agile Software Development with Scrum, *Prentice Hall*, 2001.

[21] W. Zhang, S. Jarzabek, *Reuse without Compromising Performance: Industrial Experience from RPG Software Product Line for Mobile Devices* Lecture Notes in Computer Science, Vol 3714, pp 57-69, 2005