The Effectiveness of Software Development Instruction Through the Software Factory Method for High School Students

A novel experiential based approach for introducing high school students to technology

Clemente Izurieta, MacKenzie O'Bleness, Mike Trenk, Sharlyn Gunderson-Izurieta

Abstract

Teaching software development in environments that mimic industry practices is essential for teaching applicable real-word development skills. In addition, these delivery-based projects engage students in meaningful design work that encourages clear, sustainable code. The Software Factory has provided such projects and environment to students at Montana State University (MSU) since the 2014 academic year. This project aimed to explore the effectiveness of such instruction for high school students with limited programming experience. Students from Bozeman High School, Bozeman, Montana, were selected to work in a team with two MSU undergraduate students with the goal of creating an Android application over the course of a summer semester. In the process, these high school students were exposed to Java, sorting algorithms, version control, and software development practices in an industry setting. This experiential report describes the experiences of the team, the challenges and rewards of using this teaching method – the Software Factory – and how the program provided a real-world experience for high school students in the early stages of their computing education. In addition, after concluding two projects, the latter of which is described in this manuscript, the Software Factory staff plans to continue to reach out to high school students, and has been approached by four private high tech companies, and two startup efforts. The Software Factory complements the demand generation strategies program in the Computer Science Department by providing a unique approach to outreach. The goal of demand generation strategies is to promote and increase enrollment in computing-related career fields at higher education institutions in Montana. Although this is a work in progress, the outcomes of the Software Factory approach as it relates to K-12 students are demonstrable and have surpassed expectations. The high school students were excited about programming in the context of a real world setting, presented and were the subject of a Q&A session at a graduate level seminar, produced a working prototype of an Android application, and one of the participating students is now enrolled in computer science at Montana State University. The participating high school will select new students to participate in the summer of 2016.

Introduction

With the demand for skilled software developers rapidly growing, it is more important than ever to ensure students are taught in *authentic software development environments*, in order to provide them with skills that will directly transfer to a software engineering workplace, as well as get them excited about professional workspaces. This kind of instruction helps to counteract negative stereotypes that students might perceive about working in the software development industry. These stereotypes can often include a perception that programming work is done alone and in impersonal corporate environments, despite the team-based, modern environments in

which most software development now takes place. These stereotypes are especially harmful to young students, who may feel disinclined to pursue a career that carries such a negative connotation. We investigate a current and successful program –the Software Factory approach with existing undergraduates, and apply it to K-12 students. The goals of this exploratory case study were to counteract negative stereotypes by

1. Having K-12 students work in a team that resembled a small professional software development group, and
2. Having students work in the Software Factory –an especially designed physical space created to promote a realistic open and modern work environment.

This case study aimed to address both goals through a summer project that involved three Bozeman high school students, two MSU undergraduate students, one professional staff, and one computer science professor, in a group software development project. Although this is a small number of students, it is representative of a small and realistic software engineering company's team size in a real world setting. Numerous studies have been conducted to characterize the optimal team size [12, 13], in Agile environments [14] and the consensus seems to be that a size of three to seven is an optimal number. This number is also an agreed upon measure in teams that focus on the design prior to the coding phase of the lifecycle[17]. Further, as a *work in progress* project, our goal was to devote as much attention to the K-12 students as possible.

This paper is organized as follows. We initially provide an overview of the Software Factory approach that is used with selected K-12 students. We then provide an overview of the case study, followed by descriptions of the case study phases –selection, instruction and implementation. We then describe the outreach component and the legal considerations when working with external partners. We conclude with outcomes, address threats to validity, and address future improvements to include additional K-12 students.

The Software Factory

The Software Factory is a pedagogical laboratory under the Software Engineering Laboratory in the Computer Science (CS) Department at MSU, and is an educational facility for undergraduate students designed for seeding entrepreneurship and researching technologies that have direct impact on local communities in Montana by partnering with non-profit organizations, as well as public and private high technology companies. It is a platform that provides the necessary processes and environment to deliver *real products*. It is about learning, sharing and growing entrepreneurial ideas that span the causal chain from inception to deployment, but not commercialization. The Software Factory brings together students and experienced professionals enabling unique cooperative projects that serve as incubation points for new ideas and technology innovation.

The idea of a Software Factory approach for MSU was developed by working in close collaboration with the University of Helsinki; however, methodology changes were required in order to accommodate schedules of MSU and K-12 students, as well as to address the needs of the local high tech communities.

The strategic goal of MSU's Software Factory is to establish a self-sustaining center that serves as an incubator for new technology that promotes:

1. Growth: Develop software prototypes that support new business ventures, or complement existing software products from public, private and non-profit groups,
2. Learning: Development of computer science and business students in the context of a real business environment, and
3. Sharing: Development of intensive, hands-on collaborations between companies and students (through projects) to explore the deployment of new ideas, research, and knowledge sharing.

These strategic goals are aligned with all other established and operational software factories, and open up opportunities to address *Growth* by positioning the Software Factory as an innovation hub in the state of Montana, where the interests of multiple parties can be achieved, thus creating win-win scenarios for all stakeholders involved. For example, students learn first-hand how to operate in a business setting with an agile/lean approach to delivering a product – *hands-on-entrepreneurship*. A company benefits by creating a pipeline of ready-to-go potential employees, as well as benefiting from proof-of-concept software prototypes that are ready for commercialization. The *Learning* component empowers students to accumulate expertise and entrepreneurial skills while companies can share their processes and techniques that force students to use real and authentic practices beyond the classroom. Finally the *Sharing* component facilitates an exchange of best practices and lessons learned that help refine future skills and competencies. In Table 1 we list the non K-12 projects that have used the Software Factory approach so far.

Table 1. Software Factory projects involving undergraduate students

Project Name	**Number of Students**	**Partner Company**	**Semesters**
Zoot Event Monitor	4	Zoot Enterprises	Fall '14 – Spring '15
Spectrum Analyzer	5	S2 Corporation	Fall '15 – Spring '16
Smart Gifting	5	Printing For Less	Fall '15 – Spring '16
Share a Ride	2	ShareLift LLC	Fall '15 – Spring '16

Project Description

To address our goals of evaluating the Software Factory as a viable vehicle to instruct and excite K-12 students about computer science, we devised a plan to recruit three high school students with interest in programming, but not necessarily extensive technical experience, and work with them as a team to develop an educational Android application describing and animating simple sorting algorithms. An Android application was chosen as a deliverable due to the prevalence of smartphones and the benefit of having an end-product that students could carry with them everywhere and share with friends. The sorting theme was chosen over other ideas, such as making a game, because it required an understanding of algorithms in addition to

Android development, and thus provided a good mixture of theoretical algorithm design and practical programming.

Throughout the summer-long project, the students would be informally "taught" by two senior computer science undergraduate students, but would also learn through implementation, pair programming, and self-directed online research (e.g., reading articles from stackoverflow[16]). This combination was intended to provide support for the students while maintaining a more independent, industry-like environment than a traditional classroom style.

Project Location

The project took place at MSU's new Software Factory[2]. MSU's Software Factory is modeled after University of Helsinki's laboratory of the same name[3], and aims to collaborate and deliver products to industry partners[1]. In turn, this creates a platform for students to experience software development in an authentic industry environment with real-world projects, problems, and deadlines. Previously, the Software Factory had only hosted teams of senior university level students as an interdisciplinary capstone course. The physical environment of the Software Factory made it an obvious choice to provide the students with a pleasant and realistic environment for the project. Figure 1 shows students working with a customer (an industry partner).

Figure 1. Software Factory Physical Space

Unlike traditional academic environments with rows of computers under bright florescent lighting, the Software Factory has a modern design including individual workstations arranged to facilitate collaboration, lounge areas, and artistic design. Not only did this create a more enjoyable place to work and learn, it also helped to break down assumptions about computer science workplaces being impersonal, overly corporate, or isolating.

Case Study

The project was divided into three main phases: selection, instruction, and implementation. The selection phase lasted from May until mid-June to facilitate the high school academic year, as well as allow time for students to be contacted, invited to meet the MSU undergraduate students and staff, and to tour the Software Factory. The instruction phase took place during the second half of June, and continued informally into the implementation phase. The implementation phase lasted from July until the end of August, concluding shortly before the beginning of the academic school year.

Selection Phase

When starting a project there is a competitive selection criterion for students wishing to participate. This is necessary in order to match skills and interest to a project. Selection criteria vary, and for this case study two undergraduate students, one of which had previously participated in a Software Factory project with a local high technology company, participated in the role of stakeholders (and mentors). Further, both undergraduate students in cooperation with MSU's Computer Science Department Outreach Coordinator and a Bozeman High School teacher helped select three Bozeman high school participants.

Once students were selected, clear project goals and general responsibilities needed to be allocated; however, and since this is meant to emulate a startup environment, responsibilities were not described in detail, with the expectation that team members would help define the roles as the project matured. Initial roles were described only to avoid the potential for overlapping areas, but in general all participating students wanted to be developers and contribute to the engineering of the product. This was not entirely unexpected, as high school students were not expected to be well versed in the varying roles of software projects.

The project's student selection criteria prioritized students based on their interest in computing, as opposed to their existing skill level. We believed that de-emphasizing existing skill level in favor of motivation to learn and interest in the field would lower barriers to students who may have developed an interest in computing later or were interested but unsure about computing as a career. This is especially important for encouraging the participation of underrepresented demographics that may be more affected by the negative stereotype of computer scientists. Our initial pool of students was selected from the Bozeman High School's *Joy and Beauty of Computing* class. The class was initially held in conjunction with MSU's Computer Science department, and covers basic programming concepts, such as conditionals, loops, and functions, using Python[4] as a language. Project students were expected to have at least some experience with these concepts.

The instructor of the *Joy and Beauty of Computing* class was asked to inform her students about the Software Factory project, and pass on the names of interested students. She referred one of her students, a graduating senior with previous Android development experience enrolled to begin in the MSU Computer Science program in the fall of 2015, as well as a junior student who had not taken the *Joy and Beauty of Computing* course but was interested in the project.

Despite originally only planning for two students, a third interested student –a sophomore with experience in Python, JavaScript, and HTML was recruited from the Gallatin Girls Coding Club[5]. While the other two students had already been selected, this student was also deemed a good fit, and the team decided to expand the project to include three students. This was a good decision because it helped with the distribution of tasks.

Ultimately, some of the decisions that were made in the recruitment and selection process created a collection of challenges. A lack of common skill level between the students made group instruction difficult, as students either felt bored or overwhelmed. Student travel schedules further exacerbated this issue, as one student with the least technical experience missed some group work sessions due to travel, causing the student to fall behind. Eventually group instruction was modified to better fit a group of disparate experience, but this could not fully bridge the skill gap between students.

In addition, the use of opt-in recruitment instead of direct contact of potentially well fitted students may have limited the number of potential participants, as students with less computing confidence may have felt discouraged from applying despite their interest or skill. Our third student, despite being well qualified for the project, did not feel comfortable asking to be considered; however, when directly asked if interested, the student was quite excited to participate.

Instruction Phase

In a traditional Software Factory setting, students are expected to have a knowledge base commensurate with a strong computer science background at a senior university student level, and thus, expected to be highly resourceful and self-directed. At the University of Helsinki students undergo an interview process for selection into a Software Factory project, and at MSU, only highly capable students undertaking the interdisciplinary option of their capstone projects are eligible to participate and are selected in cooperation with the stakeholders of the project. In our case study, we were aware of our constraints of working with high school students, and introduced an instruction phase meant to help bridge a knowledge gap and to instill confidence in their ability to tackle new and difficult concepts. Furthermore, the instruction phase served to break the ice with the undergraduate students and staff.

The instruction phase took place over several weeks and was designed to provide a common base of knowledge for the students in topics such as Java, Git and Github, sorting algorithms, and basic object oriented design. While the student's education in these topics continued throughout the implementation phase, dedicated time for instruction and skill building was required before app development could begin as most of these topics were not taught in high school. In order to counteract the large gap in technical experience, group instruction was modified from traditional group lectures to be a brief collective introduction to a topic, followed by individual student work where the mentors provided additional instruction or assistance to students as necessary. This allowed students familiar with the basic concepts to explore more challenging aspects while giving the less experienced students a chance to work one on one with the student mentors. In addition, students were also given tutorials to complete outside of group

work time to give the students common backgrounds for lessons and make group time more productive.

Students were initially introduced to Java through a cursory discussion of its similarities and differences to Python, the common language of the students. Next, the team began discussing sorting algorithms and was asked how to sort a list of integers, initially a list of numbers on paper. This procedure allowed students to familiarize themselves with the problem using visual cues. Once they identified an algorithm, they individually worked on implementing it in Java. The students were taught Bubble, Selection, and Insertion sort through this method, and the code they wrote served as the core sorting code of the application.

Simultaneously, the students were also exposed to version control. After a brief discussion of the importance of version control in software development teams, students were given a list of general steps for using Git and Github to push and pull updates to code bases. They then practiced these steps by sharing each sorting method implementation. This practice became invaluable as development on the application began, and students worked on separate features in sometimes colliding files.

Implementation Phase

The implementation phase consisted of design cycles, technical research, and implementation. During this phase, the team met three times a week for group sessions. There were two, two-hour weekday sessions, followed by one three-hour weekend session. Together, the students and mentors created an initial design at the beginning of the phase; which served as a wireframe for the project. The basic framework of the application, consisting of a navigation drawer and an example fragment, as well as example code from an existing Android application was provided for the students as a reference. Periodically throughout the phase, the team would evaluate progress, determine potential technical blockers, and adjust the design accordingly.

A Software Factory has no advisory board, and no organizational chart –they are small and meant to emulate a startup environment. All students participating in a given project are peers; however there is an experienced coach to help facilitate the lifecycle of a project. In our case study the MSU undergraduate students played the role of mentors. The lifecycles to develop successful prototypes use *Agile* methods such as Scrum, XP, Kanban[6,7], or a hybrid. Faculty from business and computer science can be involved in the selection of potential projects. The preferred method used in the Software Factory for tracking progress continuously while at the same time providing instant information to stakeholders, uses a method that borrows from Scrum practices and the visual aspects of Kanban. This approach is called Scrumban[10,11]. Kanban is a scheduling system that allows the tracking of multiple tasks as they move from one stage of development to another. Software engineering teams have adopted this industrial approach (originally used in the automotive production plants of Toyota) to track deliverables of components. The Kanban board is the central component of the system because it allows all interested parties to visualize progress. Various software products exist that have enhanced this experience by tracking a number of project team metrics such as *number of tasks*, *project speed*, *tasks per developer*, as well as with predictive capabilities.

The original approach for dividing work and tracking progress was to use the Scrumban/Kanban method. It was intended that each team member would move through the Kanban board, assigning himself or herself to a task, completing it, and taking a new task as they progressed. However, it soon became apparent that the tasks were difficult to break into pieces that were both meaningful and within the skill level of the students. Students often became stuck on their task, slowing development significantly, creating concerns about meeting the project deadline. Instead, the team complemented the original approach by adopting a pair programming driven approach for the more difficult tasks. The students would pair with a mentor when necessary, working together to solve problems as they occurred. In this way, students were able to gain problem-solving experience and programming insight from the mentors, avoid getting stuck for too long on any one task, and still be the primary force in solving difficult problems. This was a key difference with traditional software factory projects where participating students can pair up, but that process is left up to the students themselves. For high school students, without knowledge maturity, adopting pair programming was a differentiator in making the project succeed.

Mentors were present during the implementation phase of the project as equal members of the team –not as instructors. This allowed the high school students to feel like they were part of the team, also influencing direction, because the mentors were seen as colleagues. As a result, there were times a mentor did not have a direct answer to a question. This lack of a "lesson plan" both encouraged students to research their problems, either by themselves or in conjunction with a mentor, and sometimes frustrated them when they encountered a stubborn problem.

By departing from a traditional classroom structure, the learning environment became much more variable. Problems often didn't have a clear solution and required exploration from the entire team. Some team members used a desktop provided by the Software Factory for the project, making work outside of group sessions difficult and limiting the ability of the team to have individual work sessions. Team members often used a different OS, emulator, or physical Android device than their peers, making development environment issues frequent and difficult to solve. The students found these issues, unfortunately part of the reality of developing across different environments for different devices, particularly frustrating due to the difficulty of determining their cause.

Outreach

The Software Factory is a powerful outreach tool. After concluding two projects, the later of which was described in this manuscript, the Software Factory staff has been approached by four private high tech companies, and two startup efforts, which have led to an additional three projects that are currently being executed. Further we have established collaborations with third parties that have supported this effort by contributing licensing of software –Kanbanize[8], expertise for taking a software prototype to commercialization stages, further high school outreach, and financial support –Zoot Enterprises[9].

In response to the increasing demand for highly skilled computer science graduates, the Computer Science Department at MSU committed to an outreach program, or demand

generation strategy. The goal is to promote and increase enrollment in computing related career fields at higher education institutions in Montana to provide skilled graduates that are currently in high demand. The outreach targets Montana K-12 schools through a robotics program, teacher training, and educational tools such as the Software Factory provided to teachers.

The Software Factory complements the demand generation strategies by providing a unique approach to outreach. As described above, the Software Factory provides a program that not only provides outreach to high school students, but also provides the Computer Science Department with a visible profile in the local high tech community. This visibility in the community provides and aids students enrolled in Computer Science degree program with the opportunity to develop the necessary skills to enter into the local high tech sector with highly paid careers options.

Legal Considerations

Working with many potential players implies the need to address how to handle certain privacy rights. The Software Factory is a pedagogical tool and has no interests in intellectual property associated with stakeholders. In general, rules are established to:

- Tag data, products and processes as private, confidential, or public
- Understand collection, handling and dissemination of data
- Obtain explicit written consent from all participants in the Software Factory (students and stakeholders)
- Establish legal agreements with authors to transfer the rights to the university when appropriate
- Create draft proposals for every project that include a research agreement, an agreement on transfer of intellectual property, and data file descriptions
  After completion of projects, and depending on the stakeholder, data and software elements are either archived or destroyed.

Conclusions

Although the Software Factory approach is a proven tool for senior students in a computer science program, this pilot study has provided us with significant insights on how the Software Factory can be improved and adapted to fit K-12 students. It is an effective tool that deems further study. Our post assessment of the study indicated that in general, students learned how to apply their limited computing knowledge in a real problem solving setting and in conjunction with team members. This was a skill that was lacking prior to their participation, and together with a realistic physical space, it served to allay the negative stereotyping of computer science.

The Software Factory approach is highly flexible and can be easily adapted or transferred to other educational environments. This is important because different organizations have different requirements with regards to the number of hours students can spend in a given project. There are also differences in geography where some organizations may have better access to stakeholders willing to participate. Further, differences in available resources can also be

significant. In fact, the original approach used by the University of Helsinki was modified to fit within the constraints of Montana State University. Our Software Factory required us to adapt by adding more flexibility in student schedules, required that we compromise to use a smaller physical space, and forced us to be creative with a shoe-string budget to equip and decorate a room to make it look as close as possible to a real work environment. As a *work in progress*, we will continue to experiment to find a formula that fits K-12 students, and have already identified changes for the next group of students that will participate in the summer semester of 2016. In the summer of 2016 we will again engage two senior computer science undergraduates, two new high school students, and two returning high school students. Bringing back two high school students (at their request) is evidence of the success of this approach, but more importantly it provides a bridge between computer science students and brand new participants because they will also act as mentors. It is important to also note, that our goals also include the participation of business students, and domain experts from other rubrics. The latter is clearly a limitation for high school students, but with some creativeness, high school students could be given different roles, thus directing them to a choice of different majors when considering their tertiary studies.

Specific Outcomes

At the close of the project, all three high school students had made significant strides in understanding version control, Android application development, Java, and software development processes. The team successfully delivered a working application, and the students reported that the overall experience was both educational, interesting and has contributed towards their interest in continuing their education in computer science. Figure 2 depicts a screen shot of the algorithm animation tool that was delivered as a final product.

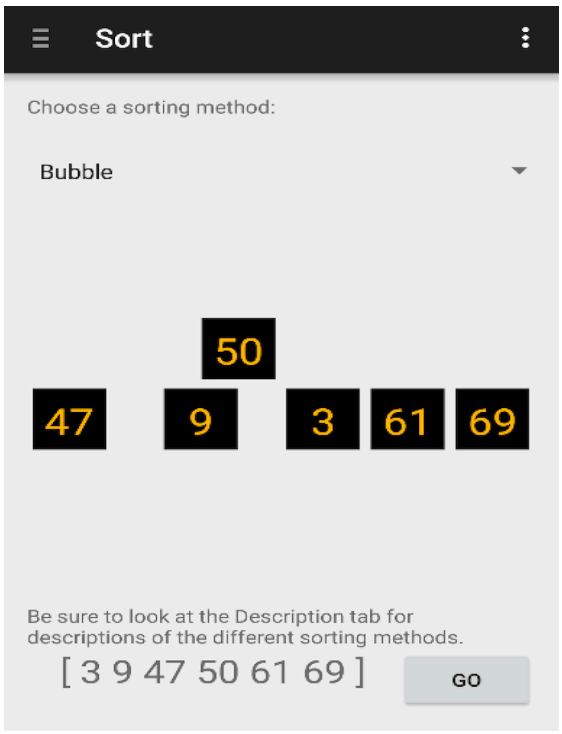

Figure 2. Software Animation Mobile Application

Recommendations for Future Projects

There were several important lessons learned about experiential projects for students early in their computer science education.

First, great care should be taken when selecting students to ensure not only that the students are a good fit for the given project, but also that they are a good fit for each other. The skill level gap between our students limited some opportunities for group instruction, made certain tasks in the project only feasible for a single student, and made true "peer" programming difficult. Given a more homogeneous set of skills between the students, instruction would be more easily structured to ensure that all students had ample time to grasp the concepts, the deliverable could have been tailored to be within the grasp of all students without boring others, and students would have been able to work together as similarly skilled individuals tackling a challenge together. The ability to work more closely with their fellow students instead of just the mentors was something the students felt could be improved in this project.

Second, while the freeform nature of the projects was in many ways helpful, some reintroduction of structure to the Software Factory may be helpful for certain groups. Specifically, the amount of structure present in the instruction and implementation phases should be based on the team's programing knowledge at the beginning of the project. Students with a better foundation can better handle a freeform system and independent work, but students without that foundation may require more structure and group work sessions in order to feel comfortable.

Finally, despite these challenges, we consider the project to be hugely successful in meeting its educational and recruitment goals. With the knowledge of problems to avoid, we believe that the next iteration of this kind of project will be even more valuable to students.

Threats to Validity

The small number of high school students that participated in this project constitutes a threat to the external validity of the study[15]. However, this is a *work in progress*, and the Software Factory technique is a proven method with undergraduate students already. At Montana State University we have had 15 undergraduate students participate in successful projects through the second year of its running, and at the University of Helsinki (our original collaborator), this program has been running since 2010 with numerous success stories[3].

Acknowledgements

The authors wish to thank the Bozeman High School students –Jessica Jorgenson, James Jacobs, and Zach Hansen, as well as their high school teacher Kerri Cobb. The authors also acknowledge the support of the Undergraduate Scholars Program (USP) at Montana State University for providing financial support for this study and Dr. John Paxton, the Chair of the Computer Science Department for his continued support.

References

[1] Fagerholm, F., Oza, N., and Munch, J. 2013. A Platform for Teaching Applied Distributed Software Development. In Proceedings of the 3rd International Workshop on Collaborative Teaching of Globally Distributed Software Development CTGDSD'13 (San Francisco, CA, USA, May 25, 2013). DOI=http://doi.ieeecomputersociety.org/10.1109/CTGDSD.2013.6635237

[2] Montana State University Software Factory http://www.bobcatsoftwarefactory.com/

[3] The University of Helsinki Software Factory http://www.softwarefactory.cc/

[4] The Python Programming Language https://docs.python.org/2/reference/

[5] Gallatin Girls Coding Club http://www.theconnectory.org/program/gallatin-girls-coding-club

[6] Raymond, L. (2006). Custom Kanban: Designing the System to Meet the Needs of Your Environment. University Park, IL: Productivity Press. ISBN 978-1-56327-345-2.

[7] Kanban http://www.toyota-global.com/company/vision_philosophy/toyota_production_system/just-in-time.html

[8] Kanbanize https://kanbanize.com/

[9] Zoot Enterprises http://www.zootweb.com/index.html

[10] Ikonen, M., Pirinen E., Fagerholm, F., Kettunen, P. and Abrahamsson, P., On the impact of Kanban on Software Project Work: An Empirical Case Study Investigation, in the 16th IEEE International Conference of Complex Computer Systems (ICECCS), 2011, pp. 305-314.

[11] Kniberg, H., Skarin, M., Kanban and Scrum: making the most of both. USA: C4Media Inc. 2010.

[12] Putnam D., 2015. Team Size Can Be the Key to a Successful Software Project. Quantitative Software Management Inc. QSM. http://www.qsm.com/process_improvement_01.html

[13] Scrum. https://www.scrum.org/

[14] Agile. http://agilemanifesto.org/

[15] ClaesWohlin, Per Runeson, Martin Host, Magnus C Ohlsson, Bjorn Regnell, and Anders Wesslen. Experimentation in software engineering. Pages 102–104. Springer Science & Business Media, 2012.

[16] Stackoverflow. http://www.stackoverflow.com

[17] Renger M., Kolfschoten G.L., Vreede G.J. Challenges in collaborative modeling: a literature review and research agenda. International Journal of Simulation and Process Modelling (2008).