# Teaching Students Software Engineering Practices For Micro-Teams

Shweta Deshpande, Joe Bolinger, Thomas D. Lynch, Michael Herold, Rajiv Ramnath, Jayashree Ramanathan
The Ohio State University, deshpande.36@osu.edu, bolinger@cse.ohio-state.edu, lynch.268@osu.edu, herold@cse.ohio-state.edu, ramnath@cse.ohio-state.edu, jayram@cse.ohio-state.edu

***Abstract*** **- Standard methodologies which have been developed for large software development teams, and Agile practices developed for small teams, make up the software engineering practices taught in the Computer Science classroom. However, given the sheer prevalence of micro teams doing business-critical software development in the field, software development best practices for micro teams must be incorporated into the software curriculum. To this end, we created a multiple-case case study (comprising five micro team projects) showing how micro teams handle the software development process. Through each of these projects, we seek to showcase what practices in existing software development methodologies, are undertaken by the developers of the projects, to achieve similar ends as developers in large teams. Specifically, the case study highlights how existing software development methodologies need to be modified, adapted and extended for micro teams. The case study and micro team guidelines were presented to students in a software engineering class within the Computer Science department at a large R1 university. The teaching was assessed using a mix of surveys and structured interviews. Initial evaluations show promise. Students were positively inclined to accept the lessons, and showed good recall of the concepts taught in tests.**

*Index Terms* - Software engineering, micro teams, case study

## INTRODUCTION

The software crisis in the 1960s was characterized by rapid increase in complexity and computing power of hardware, and difficulty in writing software utilizing those capabilities. Often, projects were over budget, late, unmanageable, and software did not fully meet requirements, was of low quality, inefficient, failing, and code was not maintainable.

To deal with the crisis, software engineering techniques were introduced – problems associated with software development did not go away, but, over the decades, more success stories than failures were observed due to the years of developing and practicing software engineering techniques [1].

What everybody agreed on, though, was that there is no silver bullet [2] – no single technology or project management approach to prevent all problems.

One of the very widely used software engineering techniques is the software development process. A software development process is basically a structure imposed on the development of a software product. Software development processes (or software life cycle models) are "used as guidelines or frameworks to organize and structure how software development activities should be performed, and in what order"[3]. These processes are "intuitive or well reasoned", and are used to package the development tasks and techniques for using a given set of software engineering tools or environment during a development project. According to Watts Humphrey, as software complexity increases in scale, a "structured and disciplined" approach to software development becomes essential for effective and successful development of software. He says that if details in software development are not managed, not even the best people can be productive. Unmanaged software development leads to endless hours of repetitively solving technically trivial problems, and time consumed by mountains of uncontrolled detail. Thus, people need the support of an orderly process to do efficient work, and as a result, a number of different software development processes have been developed, practiced and widely accepted over time [4].

An important aspect related to software development is the people involved in it – the size of the development team. We usually see that a lot of the software development methodologies handle development activities for software teams of varying sizes – large (> 25 developers), medium (10 – 25 developers) or small (3-10 developers) [5]. But studies show that a significant number of software projects are done by *micro* teams, often with just a single developer [6]. There are no software development methodologies that target such micro teams, and using existing methodologies – meant for the larger teams – with micro teams, leads to gaps in implementation. There is, hence, a need to define a framework that will provide guidelines to micro teams in developing software.

Teaching students software engineering is an important step in getting them ready to do software development in the real world. As students, they do a number of projects (often individually) to understand and master computer science fundamentals and programming techniques, as well as

software engineering techniques. Since more often than not they will come across projects involving micro-teams, it becomes important to include micro-team software development techniques in teaching software development (or applied software engineering) to students [7].

## RELATED RESEARCH

At present, a number of methodologies, accepted as standard, can be divided into two broad categories: *traditional* and *agile*. Traditional methodologies are characterized by a predictive approach, comprehensive documentation, and being process-oriented and tool-oriented. Examples of this category include the waterfall model (along with its variations), spiral model, and V-model. Agile methodologies, on the other hand, are characterized by people-orientation, adaptiveness, conformance to actual requirements, a balance between flexibility and planning, an empirical process, a decentralized approach, simplicity, collaboration and small self-organizing teams. Examples of this category include extreme programming, Scrum, and RUP.

### *A. Teaching Software Engineering in the Classroom*

A lot of effort has been put in to teach software development methodologies in the classroom, either through specialized software engineering courses or through Capstone courses by integrating all aspects of software – programming/hardware/design). Previous work includes bringing XP to the classroom [8]-[10], and teaching traditional and agile techniques "in-the-small" [11]. Even then, if the idea behind these courses is to bring the real world into the classroom, these practices don't adequately address micro-teams. Capstone courses have also been designed: from teaching agile methodologies in the classroom by lecturers using real projects [12], to industry-academia collaborations in teaching software engineering [13]. Again, these do not truly capture micro-teams – most courses consist of projects done in teams of around four developers. Rarely, if at all, are capstone projects done by single developers.

### *B. Software Methodologies and Team Sizes*

Of the existing software methodologies, generally traditional ones are more suited to large development teams, while the agile ones are geared towards medium to small teams. Apart from the large/medium/small teams, though, micro-teams have neither been clearly defined, nor are there any methodologies focusing on them. Separate formal and informal studies, as well as general observations and experiences of software developers have proved that software development does not always occur in large, medium or small teams [3].

### *C. Definition of a Micro-Team*

Our definition of a micro team as a team has two dimensions – quantitative and qualitative. The quantitative characteristics consist of the number and roles of persons involved, while the qualitative characteristics consist of the effects of the quantitative ones.

**Quantitative Characteristics:**

- The team consists of not more than 3 – 4 people actively involved.
- There is at least and only one developer programming full time.
- The rest of the team has any of the following roles: a business analyst, a project manager, a surrogate customer, or a technical advisor.

**Qualitative Characteristics:**

- The team is constrained by one developer's knowledge and perspective.
- The developer becomes single point of failure.
- Multi-person practices don't work.
- Peer review is not possible.

A 2005 study published by the software consultancy firm Quantitative Software Management shows that over 50% projects are done by teams sized 1.5 – 3 [14]. In another 2009 study by Scott Ambler, more than a third of all projects had team size 1 – 5 [15]. In yet another study made by the author, a number of projects from the Free Software Foundation were analyzed, of which over 80% projects were developed by single developers [3].

The presence of micro-team projects on such a large scale, along with the absence of any software development frameworks dealing with them, highlights the need to create an exclusive and complete software development framework for micro teams that can lead to efficient software development.

## LESSONS FROM THE CASE STUDY

Our case study included five micro-team projects:

- *Calendar*: application integrating shared web calendar, list functionality and maps
- *Sensor Cloud*: cloud application controlling wireless sensor networks through Internet
- *Website*: website for a company
- *Health Survey*: web-based survey for heart patients
- *Complex Flow Analysis*: Application collecting data and doing mathematical calculations

It brought forth a number of insights into the micro-team software practices that were then taught to students as part of the software engineering course. The learnings are summarized below:

A. ***Workbook-oriented, intra-team communication***: The workbook approach works well for micro-teams for communication and progress tracking.
B. ***Focus on developer-specific documentation***: Projects invariably get transferred from one developer to

another, either within a team or between teams. In such a case, it is important to create developer's documentation.

C. ***Work products specification based on stakeholders***: It is a good idea to loosely draw up a list of work products to be created that would best suit each team's composition and type and number of customers; this enables the team to structure their process around those work products.

D. ***Offsetting new developer costs***: Since costs of adding/replacing developers are very high, ways to offset these costs need to be thought of.

E. ***Use of subject matter / technical experts***: If the micro team does not already have a technical advisor, means of getting access to one from outside should be available.

F. ***Technical design of solution***: Ways to deal with the constraints imposed by the knowledge of a single developer need to be found in order to enable better informed decisions on technical designs.

G. ***Style of development process***: Again, since it is not always known whether more power lies with the customer or the development team, ways to deal with both situations should be come up with beforehand to minimize unanticipated situations.

H. ***Ease knowledge transfer process***: Knowledge transfer, one of the hardest problems in micro team environments, should be addressed.

In the following subsections, we will discuss these learnings in greater depth.

*A. Intra-team Communication*

In three of the five projects in the case study, the development team used an internal workspace to record the progress of the project. Items recorded include: requirements, bugs, feedback from the customer and other stakeholders, work products created in the process, and other issues. The workbook approach served as a communication space for the team and also ensured that everyone on the development team was using consistent terminology and had the same understanding about each issue.

In contrast, the agile approach uses few more formalized tools such as story charts, and pair programming between developers for communication within the team. Structured approaches use heavy documentation for communication; several documents are created in each development phase.

*B. Developer-Specific Documentation*

Often when a single developer is developing a project, the emphasis on creating developer-specific documentation is minimal. As opposed to this, when multiple developers are developing a project, communication and coordination between developers warrants creation of developer-specific documentation. This lack of adequate documentation was observed in four of the five projects we studied. Problems were caused when these projects were handed over to the next developer and they had no information about where the project was in terms of solution implemented except for the source code.

Similar to the micro team approach, agile teams produce very light documentation. They only produce as much as required, which is more than micro-team documentation. Instead, agile approaches use practices such as pair programming and team co-location. In contrast, structured approaches make use of very heavy documentation.

*C. Number and Types of Work Products*

The scope of the project and the number of stakeholders involved in it determined how many work products were created, and of what type. We found that the more the number of stakeholders, more the number of work products was produced [3]. The types of stakeholders also affected the kinds of work products produced. For instance,

- Presence of more non-technical customers produced user stories documents (as in Calendar).
- Presence of technical customers produced more formal documents like requirements document, design document (as in Cloud Sensor).
- Presence of only a single developer stakeholder in the team meant that developer-related work products (i.e. technical documentation) were never created.

The work products created in agile approaches also depend on stakeholders, but some work products are always produced – story cards, iteration charts, etc. The work products created in structured processes, on the other hand, are almost always fixed in number and type – only a few work products, depending on the application type, may differ in type and number.

*D. Cost to Add/Replace New Developers*

The cost of adding a new developer is very high for teams with only single developers in comparison to teams with more than one developer. When there is only one developer, that developer must split his or her time between training new developers and writing new code. In some cases, like when replacing a sole developer, there is no one to code at all. This situation arose in both Calendar and Health Survey projects. At the point of transition, there was no developer actively programming, with the new developers mostly just undergoing training. This brought work to a complete halt for some time in both the projects. The cost increased, then, in terms of no new value being generated in terms of code , because no new code was being written. This cost is relatively lower in agile and structured teams. The presence of more than one programmer ensures that development,

though slowed down for some time, does not come to a halt, and value in terms of new code is still generated.

*E. Subject Matter Experts / Technical Experts*

In micro teams, a subject matter expert (SME) or technical expert might not always be part of the team; instead, they may have to be specially brought in from the outside. In four of the five projects, there existed a technical advisor either inside the team or outside the team but related to the project. The developer's learning curve for the technology and/or project shortened and the general development process sped up due to the presence of such a person. In agile and structured teams, on the other hand, SMEs and technical experts are part of the team itself.

*F. Technical Design of Solution*

Since only a single developer worked on each project, the technical design of the solution was constrained by their knowledge. Systematic evaluation and identification of the best framework is not done in most cases since there is only one developer. For example, CakePHP was used on the Health Survey project with very little comparison to any other framework. As a result, any new developer who later replaces the previous one is stuck with this design; they have to learn it if it is not already known, and then have to work on the design, whether or not it is the best solution. Both agile and structured teams, on the other hand, have more than one developer to evaluate and design a solution; in fact, both of these types of team will, in most cases, have a special architect or group of architects to take care of the design.

*G. Development Process Style*

The style of development that the process follows in a micro team environment clearly reflects the balance of power between the customer and the team. A non-technical customer was seen to be less powerful than the micro team in two projects, and the development process reflected the micro team's style of work. However, in two other projects, a technical customer was more powerful than the micro team and had more control in how development took place. In these two projects, it was the customer who wanted stand up meetings every other day, so work had to structured taking that into account.

Composition of the team also played a part. When a majority of the team was made up of surrogate customers as opposed to other roles, the balance again tilted in the customer's favor. In one project with one developer and four customers (Complex Flow Analysis), the developer fit into the customer's schedule and meetings instead of creating his own schedule and requirements and then asking the customer to fit into that.

In contrast, agile teams reflect the team's style of working, but the process is shaped considerably by customer input. Structured teams' development processes completely reflect the team's style of work.

*H. Knowledge Transfer*

Of the five projects, four were handed over from one developer to another. Knowledge transfer in these cases proved to be a challenge because of one or more of three reasons: lack of adequate documentation, no formal knowledge transfer process followed (meaning both developers sitting together to teach and understand), or unavailability of the previous developer to answer questions. One project (Health Survey) found the knowledge transfer especially challenging because of the presence of all the above three reasons.

This process is usually relatively easier in the agile approach, as they have knowledge transfer processes that include team interaction happening through tools such as pair programming and team co-location. Since agile projects usually prefer face-to-face communication, approaches such as "rotating people in each iteration, completely replacing the team gradually" and "a developer spending at least a couple of months to work with the new team" [16] are common.

In contrast, the structured approach usually has formal knowledge transfer processes in place, with a lot of stress on documentation-based knowledge transfer. This results in a time consuming process.

In addition to these eight insights, a couple of other observations came to the fore which showed how, in a some ways, micro-teams are really not very different from their larger counterparts.

*I. Process Flexibility with Development Approaches*

Development processes of any team – whether micro or small following agile or large following structured approaches – show flexibility with respect to the development approaches that they follow. What differs, though, is the degree of flexibility seen in each. Micro team processes are seen to be the most flexible ones, closely followed by agile approaches and last of all structured processes. This flexibility depends on the team size. Smaller the team size equates to lower momentum, which means it is easier to change approaches. The opposite is true as well: the larger the team, the greater the momentum, and the more difficult it is to change approaches.

The development process followed within a micro team is usually not clearly defined, and consists of only a single developer. Hence, it is very flexible and allows different parts of the same project to be developed using different approaches.

In one of the projects (Complex Flow Analysis), the commonly followed development approach of requirements-design-implementation was followed for developing most components. For developing one specific component,

though, a test-driven development approach was more suitable, so the developer changed the development process from traditional to test driven.

Another project (Cloud Sensor), also showed this flexibility. In this project, a data storage component was developed using a test-driven approach, while a security component used the more common approach of requirements-design-implementation. This would have been more difficult in a larger team, because larger teams need more coordination, and getting out of a fixed process would be more difficult.

### *J. Customer Interaction*

In all the projects studied, the development team had weekly meetings with the customer, either over the phone or in person. Each meeting served three purposes: demonstration of the software built, discussion of other work done by development team, and requirements discussion, including both a determination of the next tasks on the to-do list and a discussion of feedback from the customer and other users. This was important because the development process had no clearly defined path, so constant validation of work done and specifying next steps was needed.

Defining long term steps and infrequent feedback on work done (e.g. feedback every month as opposed to every week) might lead to waste of time and effort in case there was either a misunderstanding of the task or requirement by a developer or a misunderstanding by the customer of what the deliverable would be.

Agile teams also have frequent and impromptu interactions with customers and considerably engage customers in the process. Structured teams, on the other hand, have interactions with customers that are much more structured and scheduled.

## TEACHING METHODOLOGY AND EVALUATION

### *Methodology*

The micro-team practices that we learned from the case study were presented to a software engineering class through one of the projects from the case study. The entire project was explained, starting from background and inception, and continuing to implementation and deployment over the course of four lectures. Micro-team practices were explained within the context of the project.

The first time, the lecturer presented the material in person to the class. A video recording was made for all four lectures. In later offerings of the course, an inverted classroom technique was used to include the material. The students were provided with the videos of the material and asked to watch them offline. After watching the videos, discussions based on them took place in class.

### *Evaluation*

For evaluating whether the students understood the concept of micro-teams, our researchers conducted interviews of students who had taken the software engineering course.

## CONCLUSION

Micro-teams developing projects are frequently observed, yet none of the commonly practised software engineering practices, both traditional and agile, are geared towards them. Neither do micro-teams feature in currently taught software engineering curriculum, leaving a gap in bringing the real world software practices to classrooms. This prevalence of micro-teams is enough to necessitate studying their distinctive practices and teaching students about those practices. Our case study of such projects provided significant insights about how micro-teams handle the development process. These learnings offer a good way to bring the micro-team aspect in software engineering education of students. Encouraging reactions of students on learning about such practices confirm the need and importance of teaching them about micro-teams.

## ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under NSF CCLI Grant No. 0837555 and the CERCS IUCRC Center for Enterprise Transformation and Innovation (CETI), supported by the NSF-IUCRC Program, Grant No. 0630188

## REFERENCES

[1] Pressman, R. S., “Software Engineering: A Practitioner’s Approach”, Pg. 11

[2] Brooks, F. P., "No Silver Bullet Essence and Accidents in Software Engineering", *Computer*, 20, 4, April 1987, 10-19

[3] Scacchi, W., “Process Models in Software Engineering”, *Encyclopedia of Software Engineering, 2nd Edition*, John Wiley and Sons, Inc, New York, December 2001.

[4] Humphrey, W., “Managing the Software Process”, Addison-Wesley, 1989

[5] Ambler, S., “T Project Success Rates by Team Size and Paradigm: Results from the July 2010 State of the IT Union Survey”, http://www.ambysoft.com/surveys/stateOfITUnion201007.html

[6] Deshpande, S., “A Study of Software Engineering Practices for Micro-Teams”, *OhioLINK ETD*, 2011

[7] Pollice, G., “Teaching Software Development vs. Software Engineering”, http://www.ibm.com/developerworks/rational/library/dec05/pollice/index.html

[8] Bergin, J., Caristi, J., Dubinsky, Y., “Teaching Software Development Methods: The Case of Extreme Programming”, *Proceedings of the 35th SIGCSE technical symposium on Computer science education*, March 2004, Vol 36, Issue 1

[9] Goldman, A., Kon, F., Silva, P., J., S., “Being Extreme in the Classroom: Experiences Teaching XP”, *Journal of the Brazilian Computer Society*, Nov 2004, Vol 10, No. 2

[10] Cleland, S., “Agility in the classroom: Using Agile Development Methods to foster team work and adaptability amongst undergraduate programmers”, *Proceedings of the 16th Annual NACCQ*, July 2003

[11] Kessler, R., Dykman, N., "Integrating Traditional and Agile Processes in the Classroom", *Proceedings of the 38th SIGCSE technical symposium on Computer science education*, March 2007, Vol 39, Issue 1

[12] Lu, B., DeClue, T., "Teaching Agile Methodology in a Software Engineering Capstone Course", *Journal of Computing Sciences in Colleges*, May 2011, Vol 26, Issue 5

[13] Rusu, A., Swenson, M., "An Industry-Academia Team-Teaching Case Study for Software Engineering Capstone Courses", *38th Frontiers in Education Conference*, 2008

[14] Putnam, D., "Team Size Can Be the Key to a Successful Software Project", http://www.qsm.com/process_01.html

[15] Ambler, S., "Agile Practices Survey Results: July 2009", http://www.ambysoft.com/surveys/practices2009.html

[16] Fowler, M., "AgileHandover", http://martinfowler.com/bliki/AgileHandover.html

## AUTHOR INFORMATION

**Shweta Deshpande** Graduate Student, Department of Computer Science and Engineering, The Ohio State University, deshpande.36@osu.edu

**Joe Bolinger,** Ph. D. Candidate, Department of Computer Science and Engineering, The Ohio State University, bolinger@cse.ohio-state.edu

**Thomas D. Lynch,** Ph. D. Student, Department of Computer Science and Engineering, The Ohio State University, lynch.268@osu.edu

**Michael Herold**, Ph. D. Student, Department of Computer Science and Engineering, The Ohio State University, herold@cse.ohio-state.edu

**Rajiv Ramnath**, Director, CETI, Associate Professor of Practice, Department of Computer Science and Engineering, The Ohio State University, ramnath@cse.ohio-state.edu

**Jayashree Ramanathan**, Director, CETI, Associate Research Professor, Department of Computer Science and Engineering, The Ohio State University, jayram@cse.ohio-state.edu