Collaborating on Mobile App Design through Pair Programming

A Practice-Oriented Approach Overview and Expert Review

Mohammed Seyam
Department of Computer Science
Virginia Tech
Blacksburg, VA
seyam@vt.edu

Scott McCrickard
Department of Computer Science
Virginia Tech
Blacksburg, VA
mccricks@cs.vt.edu

Abstract—In many of today's small companies and startups, developers may not give enough attention to the importance of UX/UI design for the product under development. Moreover, such small teams may lack the required organization skills that help them to collaborate and work together. As a result, such teams usually face problems with delivering the "right" product, as well as not being able to follow a sustainable development process. In a similar context, CS students in programming classes are facing almost the same problems of small teams in industry. To tackle these problems, various approaches for integrating agile development methods and UX design methods have been proposed to help developers carefully consider the UX requirements, and to be able to organize their work environment. In this paper, we explore how such integration can be a good fit for both CS students and software developers, especially for those who work on mobile development. We present our proposed set of integration guidelines, and then we focus on Pair Programming (PP) as an agile practice that promotes a collaborative work environment. The expert reviews conducted in this paper helped us exlore how PP can be introduced to CS students (and later, to developers in the market) to support collaborative work environments. Moreover, Using PP in class provides an adaptive and collaborative teaching approach that can be used in CS programming labs. We also discuss the pitfalls that can affect developers during PP sessions, and how to avoid such negative effects. We conclude by providing a set of recommended practices that can be applied to introduce PP for developers in both academic and industrial contexts.

Keywords—Frameworks for collaboration; collaborative software development; collaboration in education; agile methods; pair programming; UX design

I. INTRODUCTION

Software developers in small teams and startups and CS students in programming classes basically share two common problems: (1) they are usually concerned with how to build the required product (the lab assignment or the course project in the case of students), with less focus on how they approach the actual customer non-functional requirements (that are mostly User Interface (UI) requirements), and (2) they don't give enough time to decide how they are going to work on their project, and they usually work with unplanned approach with the main goal of delivering a working piece of software. To tackle these problems, developers need to pay more attention to the non-functional requirements (specially the UI requirements), and they need to follow a flexible development approach that helps them stay focused with minor planning and without hindering their progress towards delivering working software.

By definition, agile methodologies are concerned with how the software should be developed, while UI design focuses on how the end users will work with the software. Although the two aspects are following different guidelines for each to achieve their goals, integrating agile and UI design practices seem to be a promising combination that can help software developers to better handle the UI requirements as well as to follow a semi-structured development approach to help them manage their programming work.

Since collaborative development environments are very important to success of any agile team, in this paper we explore how Pair Programming (PP) can help developers collaborate effectively on their development processes. One of our research objectives is to study how a collaborative work environment can help developers build better and usable interfaces. However, for this specific paper, we focus on how we can promote a more collaborative work environment through an agile practice such as PP.

II. RELATIONSHIP BETWEEN AGILITY AND USABILITY

Agile methodologies have appeared to deal with the new problems that began to evolve with the new era of web – and then mobile – applications. For a while these were known as "lightweight" methodologies, but then the term "Agile methodologies" appeared to describe this group of new methodologies. These methodologies attempt a useful compromise between no process and too much process, providing just enough process to achieve reasonable results.

Agile methodologies have attracted a lot of attention; the main reason was that they seek to cut out inefficiency, bureaucracy, and anything that adds no value to a software product. Proponents of agile methodologies often see software specification and documentation as adding no – or minimum – value [1] . Agile methodologies are less document-oriented, usually emphasizing a smaller amount of documentation for a given task, and strongly advocate for human communication and collaboration over defined and repeatable activities as mechanisms for developing quality software [2].

What is important about agile methodologies is not only the practices they use, but also their recognition of people as the primary drivers of project success, coupled with an intense focus on effectiveness. They stress two concepts: the unforgiving honesty of working code and the effectiveness of people willing to work together [3].

Usability Engineering (UE) and User eXperience (UX) are currently considered important concepts to the software mainstream [4]. UE deals with issues such as system learnability, efficiency, memorability, errors and user satisfaction [5]. UX and its related aspects are being considered by software engineers and researchers because of the huge demand on the web, mobile, and internet applications in general. The UE and UX processes focus on developing systems that are adapted for end users.

Chamberlain and Sharp in [6] argue that although there are similarities between agile development principles and UX design guidelines, there are also differences that make it hard to combine both in a single project. The three main similarities are:

1. Both of them rely on an iterative development process, building on empirical information from previous cycles or rounds.
2. They both place an emphasis on the user, encouraging participation throughout the development process.
3. Both approaches emphasize the importance of team coherence.

On the other hand, the main differences between the two approaches according to [6] are:

1. UX advocates require certain design products to support communication with developers, while agile methods seek minimal documentation.
2. UX encourages the team to understand their users as much as possible before the product build begins, whereas agile methods are largely against an up-front period of investigation at the expense of writing code.

In [7], Ambler showed what he called "challenges" of combining agility with UX design. Four of Ambler's main challenges are:

1. Different goals: based on Lee's work in [8], Ambler argues that agile methods focus on the design, implementation, and maintenance of software systems, while overlooking the design of the human-computer interfaces through which those systems are used. On the other hand, UX designers focus on developing systems so end-users can use them effectively but do not account for the underlying system design, implementation or market-driven forces that are of most importance to software engineering.
2. Different approaches: The UX methods try to get a holistic view of user needs and come up with an overall plan for the user interface before starting implementation, while agile methods favor little up-front design and focus instead on delivering working software early.
3. Organizational challenges: The agile community follows a highly collaborative organizational strategy where teams are self-organizing, which is not the common case with UX teams. Hodgetts in [9] shows that a center for UX with a strong organizational and management hierarchy can be problematic as opposed to the fluid organizational structure that agile teams have.
4. Process impedance mismatch: The agile community is always against early detailed designs, which they refer to as Big-Design-Up-Front (BDUF). On the other hand, many within the UX community prefer more comprehensive modelling early in the project to design the user interface properly before actual development begins.

These differences basically exist because of the fact that those who invented agile methodologies are mainly programmers who focus on building working software rather than usable software. They deal with what users want (i.e. what users say they want) rather than what users need (i.e. what users will actually use the software for).

The similarities between agile development guidelines and UX design principles, together with the advantages of following the two families of methods have encouraged researchers and practitioners to find ways to combine them in their development projects. However, the differences that were presented in the same section have always been challenging for those who want to go for such combination. Some examples of such integration approaches are presented in [4], [7], [10] [11], [12], and [13].

III. Recommendations for integrating Agility and Usability

Based on our literature review, we found that the suggested integration approaches may not fit the main targets of our research, which are developers of small teams and CS students. Therefore, we proceeded by exploring how to integrate agile methods with UX design principles in a light practice-oriented approach that can be accepted, digested, and easily applied by developers.

We started working on this by surveying some of the suggested integration approaches (examples were given in the previous section). We also had a case study where practices from a more general framework (the "Tragile" Framework [14]) have been applied in a small project to demonstrate the applicability of combining practices from both agile and UX areas.

Based on our literature review, and considering the implementation of some practices on a case study, we can summarize our recommendations for applying UX principles within agile environments for small teams in the following points.

1. Although agile methods aim at satisfying customers, they need explicit practices to satisfy UX/UI requirements.

"Satisfying customers" has always been claimed to be the main objective of agile methods. Therefore, most of the proposed agile practices tend to focus on rapidly providing high-quality software that achieves user goals. However, such goals are usually related to system's functionality; giving less consideration for the non-functional requirements. Hence, providing UX-specific practices ensures that this category of requirements (i.e. the interface requirements) will have the same priority as the regular functional requirements, while still maintaining the agility of the software development process and the quality of the produced software.

2. It's usually better and easier for small teams to be given practices to apply rather than guidelines to follow.

Although agile methods are about flexibility and adaptability, it is still need to be manageable and controllable especially for the less-experienced teams. High-level guidelines are useful as introductory ideas to the agile thinking, and they give project managers and experienced developers the space to innovate by applying such guidelines on their own ways. However, guidelines can be misinterpreted or misapplied by less-experienced team members, which can lead to major development problems. Providing detailed practices encourages team members to collaborate regardless their experience level, and avoids the problems associated with doubting what the real meaning of particular guidelines could be.

3. The simpler (lighter) the practices are, the easier for developers to apply

With less-experience developers, or developers who are new to the agile world, it'd be important to introduce agile methods for them through straightforward easy-to-apply practices rather than complicated ones. Both agile and UX practices have different levels of complexity considering introducing and applying them. The case study showed that developers not only learned the simple practice faster and easier (which is expected), but they also got the sense of agile thinking through such simple practices and then they gained the required level of self-confidence for them to delve in "heavier" practices that need deep understanding of agile and UX concepts.

4. Some practices of agile development and UX design are common sense, however, they need to be planned, guided, and made sure to be applied.

Based on the developers' feedback from the case study, they were surprised that some practices that they do from time to time are actually recommended agile and UX/UI practices. However, they were not able to evaluate their performance because they were not consistent in applying such practices. Therefore, even with the simple "common" practices, planning and guidance are important so that developers performance can be measured, and to help project manager and the other team members to reflect on their own work. Such consistency in applying practices helps teams adjust their performance while progressing towards applying "heavier" practices.

Hence, the general theme of the four recommendations can be seen as: practices are always more manageable and easier to follow than high-level recommendations.

IV. Towards a Collaborative Practice-Oriented Integration Approach

Based on the above recommendations, we decided to focus on studying specific practices that are both agile and UX-oriented. One of such practices is Pair Programming (PP), which is an old technique that goes back to the mid-1950s as shown in [15]. However, PP hasn't gained the IT community attention until its revival by Kent Beck when he introduced his eXtreme Programming (XP) methodology [16].

PP, together with SCRUM daily meetings [17], have always been argued to be the most effective practices that support the agile manifesto's first principle, which values "Individuals and interactions over processes and tools" [18]. PP is also considered to be a typical application for the sixth principle of the twelve agile principles, which states: "The most efficient and effective method of conveying information to and within a development team is face-to-face conversation" [19]. Therefore, PP seems to be a reasonable choice for us to begin exploring as a practice to support collaborative software development, which will help us reach our objective of better team organization and enhanced development process.

Moreover, PP has always been studied as an agile practice; giving less consideration on how it could affect UI design. For example, [20] presented a study of using PP for teaching Human Computer Interaction (HCI) class, but their study didn't show the impact of applying PP on the UIs developed in such class. Therefore, we are extending our work to study how PP can be used to design better user interfaces through considering UI design requirements as well as the regular agile development requirements.

Unlike the previous studies that applied agile methods in classrooms, we are interested with the UX requirements as well as the regular development tasks. Moreover, we are focusing on applying certain practices that we believe to be easier to follow than the regular agile guidelines. Our proposed practices are related to how to introduce PP to students, activities that ensure that students fully comprehend the new environment, practices to gradually apply PP in both in-class and take-home assignments, and specific practices to enforce an in-class agile environment rather than the traditional class settings.

By exploring PP as a collaboration-promoting practice for software developers (either on companies or CS classes), we aim at achieving three main goals:

1. To provide CS educators with a new adaptive teaching approach that is more collaborative and student-oriented, which they can use in their programming labs instead of the traditional task assignment approaches.
2. To extend our work to be applied in industry for teams who are designing UIs for mobile devices.

3. To enhance software development process for small teams by providing practices that support collaboration and knowledge transfer among team members.

V. UNDERSTANDING IN-CLASS PP THROUGH EXPERT REVIEWS

To better understand how developers interact with each other while working in pairs, and to ensure that we have better understanding for the context of developers in a classroom setting (i.e. students), we conducted an expert review with two pairs of experienced Android developers to understand from them the potential advantages and pitfalls of this approach. One pair of experts was graduate students, and the others were undergraduates:

1. For graduate students: they are current CS PhD students who worked in the field of mobile development for long time. They've also worked as Graduate Teaching Assistants (GTAs) for the mobile development class. Therefore, they are experienced in programming in general, mobile development, and education. It was important for our research to get this level of experienced developers to practice PP so that they share their thoughts and feedback on how the process was for them, as well as what worked and what didn't work when it comes to practicing PP.
2. For undergraduate student: they are senior students in CS department, and both of them worked as Undergraduate Teaching Assistants (UTAs) for mobile development class. Their Android development experience level is lower than the graduate level, but they are experts in dealing with students in programming classes.

Based on that background, we moved forward to conduct two active walkthrough PP sessions to get the participants expert reviews about the process. We observed their interactions throughout the session, and they shared their insights during and after the sessions. We – together with the experts – agreed that the programming assignment that was to be worked on during the sessions should have some major criteria, which led us to come up with the following task requirements (categorized based on the corresponding criteria):

1. To be a familiar task: to implement a calculator that performs the basic operations and runs on android devices.
2. To be flexible: the task was open for whatever assumptions and decisions made by pairs. A calculator can be anything from two simple text boxes that use the devices keyboard to a full-sized screen with all buttons and operations.
3. To have usage context: that assignment definition stated that this calculator will be used by fourth and fifth graders (ages 8-11 years old) to introduce them to calculators and get them to be familiar with them.
4. To have no specific interface requirements: as we wanted to observe how pair will come up with interfaces that fit the required task based solely on their understanding of the usage scenarios.
5. To be time-limited: pairs should deliver their working prototype within one-hour time frame. We wanted to see how time constraints would affect interactions as well as decisions made by developers.

The experts collaboratively decided that for one-hour session, exchanging roles between driver and navigator would occur every 15 minutes, with the session facilitator working as a time-keeper. After getting seated and prepared with the required information and tools, the sessions were ready to start.

A. Session 1 (Graduate experts)

Developer A was more experienced in Android development than developer B, as he worked more time with it and taught more classes related to it than B. However, both of them were experienced developers and they defined themselves as "we don't know everything about Android programming, but we are confident that we can find solutions for the problems we face even if they are completely new to us". Although they dealt with each other before, it was their first time to work together in pair on a programming assignment.

The first decision made by them was about which IDE they should use. Both of them were experienced with Eclipse, while only B was the one familiar with the newer Android Studio platform. From that point, A had the suggestion to go with the "common ground", which the Eclipse, so that they can save time to focus on the application rather than getting to know about the new tool.

It was important to notice that before making this decision, B explained the main benefits of using the newer tool, which were unknown to A. B has even did a short demo showing some of the "nice" features provided by Android Studio. However, both of them were satisfied by working on the IDE that they both knew about. So, their decision of working on the older tool didn't prevent A from learning some features about the tool that they decided not to use. This is an important point about PP and collaborative environments, which is that developers don't only learn about what they are using, but also about the options/tools/approaches that they decided not to use.

Once developer A – the first driver – was done with setting up the new project, they stopped to talk about the layout that they should use for their application. After exchanging some verbal ideas, they hold markers and began drawing on a whiteboard right behind them. They approved using the "grid view" element although both of them didn't work with it before. They were confident about their ability to try something for the first time as long as it will achieve the required results. Therefore, they directly moved forward and began implementing their basic solution idea.

Opposite to their expectations, dealing with grid view wasn't that easy, and they had to handle some issues related to sizing, positioning, and alignment of cells. Suggestions to solve such problems came from the two developers, as they faced those problems during the first three rounds of the four-round session. Most of the times developer A's suggestions were approved and out to action even if B had some other solutions,

but that didn't cause any problems as A's suggestions were based on experience and usually provided better solutions. However, two main points affected this type of interaction:

1- Developer A suggested a line of code to be written by B, while B showed him that this might not be the right way to do it. However, A insisted that it's a good way to do it, so B just did as what A proposed and they moved forward. Starting from this point, B's level of interaction and suggestions were less than before.

2- Later when an error was found by the debugger, developer A tried to fix it by editing some parts written by B. However, they discovered that the problem aroused from the previous suggestion of A. It then became clear that B was right on the first place. Starting from that point, B's level of interaction increased as his suggestions, comments, and insights were much more than before. Moreover, A began to consider B's inputs and asked for his approval more carefully than the previous round.

These two points show the importance of self-confidence, mutual respect, and openly sharing thoughts for the success of PP sessions. For this specific session, the two experts had good levels of self-confidence and even higher level of mutual respect for each other. However, A seemed to be more confident about his abilities, which made him – to some extent – disregard B's suggestions. This style of interaction doesn't lead to a "good" PP session. However, it was smoothly corrected with B's trials to always share his thoughts and with both of them discussing what they are working on. Facing situations like the two previously-shown above could lead PP session to success or failure based on how the pair handle it.

The discussion about users and usage scenarios appeared early when the pair worked on the main layout. They began asking and answering questions about what would the users (8-11 years old kids) need to find in such application. Based on their assumptions, they decided to include only the main arithmetic operations (addition, subtraction, multiplication, and division). They excluded fractions, and that's why they didn't include the "dot" button in their final version. They planned to work on some graphics to better fit the young users but the time didn't allow for this. However, they skipped an important function, which is the ability to delete one or more digits in case of errors. That mistake appeared only on their final test before delivering the product. Although it'd be easy for them to correct that problem, it still shows that their discussions have missed some major usage scenarios. This shows that PP enhances the developers' awareness about user experience, but it still needs supporting steps to ensure that developers fully understand how users will interact with their application.

It was clear for the pair on the final round that they will not be able to deliver a fully-functional prototype, so they decided to go for at least implementing one operation (which was the addition). They skipped some interface requirements in favor of providing a prototype that works even with some errors (exceptions). On that final round, the discussion between the pair was minimized to save time, and developer B was driving the keyboard, with A's intervention only to correct mistakes or to suggest ideas that help B goes faster. At the end, they were able to provide a working Android application that can be used on an Android device to input two numbers and calculate their sum.

The first words said by experts after they were done with the session were "it was fun" and "it was exciting". It was obvious that they were practicing a game-style pairing where their team was playing against time. Even if they didn't provide the complete required product, they were able to deliver a small working version of it. The two experts declared that they enjoyed the time spent during that session. Developer A showed that it'd take him more time to work on such assignment if he was to do it on his own, because developer B used some coding practices that A would otherwise not use them. Developer B indicated that it'd take him almost the same time to do the same job, but he believed that the quality of the task coming from the pair would definitely be better than of the one he'd work on by his own. They both agreed that the one who drives the keyboard is usually in a better position to decide than the navigator. That clearly appeared on the final round where developer B decided to go for a coding practice that A didn't prefer (using hard-coded listeners), but A didn't stop him because time was running and they wanted to have a working demo. They also felt that sometimes the navigator wanted to get the keyboard to do something that would be faster than leaving it for the driver to do, but of course they didn't because it wasn't allowed during that PP session.

Developer B showed that he wasn't able to try some solutions using his own way because developer A was usually deciding on the fly while he's driving the keyboard. This obviously changed after the second round, which is related to what developer A admitted: "B is really a good programmer, I liked him. He's really better than what I thought!". This feedback made it clear that even with some pre-assumptions that can exist among developers, PP usually helps to correct some of those false assumptions after the first rounds. The two developers said that they both learnt new things from each other. Those new things were more related to coding practices, tips, and habits. They also showed that they enjoyed learning how to deal with a new layout structure (the grid layout) together, which made their learning time shorter. As for their personal feelings regarding the PP session, they concluded with almost the same sentence: "That was pretty awesome, and I would like to do that more".

B. Session 2 (Undergraduate experts)

Both developers C and D had almost the same experience level in programming in general, and in Android programming specifically. They knew each other before this session, but they never worked together on programming assignments. In assessing their experience level, they described themselves as "being able to get the required knowledge to get the job done".

Both of them were familiar with Eclipse, but only C worked with the newer Android Studio. Unlike the pair of the first session, they decided to go for Android Studio after C explained its advantages and how it made some issues easier for her. Starting from this point, almost all decisions were done the same way: one developer suggests something showing her rationale, the other approves. It was clear that this pair dealt with more peer-to-peer interaction level, rather than the leader-employee interaction model that appeared on the first round of

session 1. Therefore, it should be clear that introducing PP is better to be done with developers of similar experience levels. Once developers in certain environment get used to the concepts of PP, pairing can then done among different levels of developers, where the learning process can be more valuable and beneficial.

Developer C was the first driver as she wanted to introduce the new IDE to developer D. After a very short and general introduction, they decided to write some basic lines of code. They then stopped after minutes when they realized that they didn't agree on how the interface will look like. After spending three minutes talking about that, they hold some markers and began drawing their ideas on the whiteboard. Each of them had her own design idea, with C suggesting a very basic and simple interface, while D suggesting a more attractive interface with more features. Both of them agreed that D's idea was better, but they also agreed that C's proposal would be more feasible because of the time constraints. Once agreed on the initial components of the layout, they began implementing the code to make it functional without giving any time to discuss the positioning, look, or any details regarding the interface elements.

The transition from being driver to navigator went smoothly every time, and all the implementation decisions were easily approved by the two developers. It was noticeable that they didn't have to perform any online search for their work. They depended heavily on their previous knowledge and what they already know. That led them to be more conservative in their implementation choices, so that they don't have to face some sudden new situations that would prevent them from being able to deliver a functional prototype on time.

One of the important observations about this session is how the pair was so careful about "getting the assignment done" rather than building a usable application. They were dealing with the task as a class assignment that will be graded, without considering how users will deal with it. The sizes and locations of the text boxes, the alignment and positions of buttons, and all the aspects related to the interface were left to the last round. They wanted to make sure that users can input numbers, click the required operation, and get results to appear on the screen. At the end, they provided a functional product with a poor design that lacks some basic usability requirements.

Although the two developers claimed that they were affected with the time limits and that they would have consider usability issues if they had more time, it was clear that their interest in user experience wasn't a priority regardless time constraints. They didn't design a complete layout before coding, they didn't talk about interface components but in implementation context, they left the layout design for the last round, and they didn't consider usage scenarios.

The two experts showed that they didn't consider "designing for school kids" or "serving as an introductory calculator" as requirements of the application. Their suggestion was: "we can later add some colors and graphics to be more appealing to school students". Developer C showed that if there's something that she'd change if she would repeat that PP session, it'd be to "spend more time on design, for both program structure and interface". Both of them felt they skipped that important part of design, and they thought they'd have provided a better product if they considered design more carefully.

When it comes to interaction and communication between them, developer C liked that her partner was always talking with her, and that she wasn't the only one who talked all the time. Being able to talk and listen while coding was an important issue that the two experts emphasized its value for the success of a PP session. They also felt that they were learning together rather than learning from each other. Since both of them were of a "similar academic intelligence level" as stated by developer C, it was easy for them to express their ideas and to be sure that the partner will understand what the implementation suggestions and coding stuff.

It was important for the pair that their experience levels are close to each other. Developer C talked about her reaction if she deals with someone with more experience, showing that she easily gets intimidated in such situations, and that she gets shy and stressed, which lead her not to gain from working with the experienced developers. On the other hand, developer D had no problems in dealing with experienced developers, but her reaction would be to leave him/her do the required job, trusting that she'll be only called if she's needed to. For D, the experienced developer will be the leader who's responsible for the hard work, while she will be the assistant who will help only when required. These two different reactions to dealing with more experienced partners emphasize the importance of pairing developers of similar experience levels, especially on the first PP sessions.

The two experts agreed that for a simple assignment like that one, PP wouldn't enhance their performance nor quality, while it may does for larger projects with more requirements and sophisticated implementation issues. However, their opinion may be affected by the fact that they didn't dedicate enough time to the requirements that they should have focused on (i.e. design, usability, and usage scenarios). The effect of PP for them wasn't clear because they jumped directly into coding, which led them to miss the main and important advantages of PP that would have helped them designing a better application.

C. Collaboration pitfalls during PP sessions

Based on our literature review on PP, together with the expert reviews discussed above, we came to highlight some pitfalls that can affect developers' performance and interaction during PP sessions. The following eight issues are ranked (from higher to lower) based on the experts evaluation for which pitfall would have the most negative effect on PP sessions conducted by students. The first listed issue would cause severe problems, while the last would have the minimum negative effects.

1. Developers coming from different backgrounds

When introducing PP for the first time, it is hard to get developers to talk to each other if they didn't already have a common background to start from and move forward. This, however, can be useful with developers who are experienced and comfortable with PP, where the diverse backgrounds will add to their skills and widen the scope of their discussions about the product under development.

2. Developers with different skill/experience levels

This can work well with pairs who are familiar with PP, so that knowledge transfer can be a major benefit from practicing PP. However, for beginners in PP, it's more important to get familiar with PP through being able to talk with their partners as peers rather than as students or learners. As we showed on the first session, the lack of peer-to-peer interaction had affected the first two PP rounds until that got fixed as they progressed in coding. Having pairs with partners of different experiences levels will be required for those who are already comfortable with PP, as that enhances the learning curve, helps transferring knowledge, and puts collaborative environment into action for the benefit of the whole team.

3. Lack of planning and time management

The two pairs didn't work as time keepers during their sessions, and our objective was to allow them to focus more on their work rather than checking the time every now and then. However, they knew that they were only allowed on hour to finish their work. It was noticeable that during the two sessions none of them have mentioned anything about time remaining. They didn't have any tentative plan on how work will go on through the 1-hour time slot. That's why the two pairs faced the same problem on the final round where they had to wrap their work up to be able to deliver a working demo. Therefore, it's important to consider time management between pairs as something that they should consider early on the first PP round.

4. Jumping directly into coding without working on design

Session 2 showed how the lack of proper software design has affected the developers' ability to deliver a quality product, and it also affected their coding, debugging, and testing experience. Less-experienced developers may oversight some important aspects of software design when they get excited about trying some new approaches (such as PP). That's why it's important to direct pairs on their first PP sessions and guide them throughout the development process to make sure that they maintain the basic guidelines of software quality procedures while working with their partners.

5. Thinking about the assignment as a "task to be graded"

That problem was clear when dealing with undergraduates, who were keen to follow the problem specifications and translate the vague requirements in the safest and simplest possible ways. Since they weren't exposed to development environments other than their programming classes and projects, everything for them seemed to be a "graded task" that they should get an A in it. Therefore, they ignored any contextual issues related to the assignment, while focusing only on the clear functional requirements stated on the problem definition. Although that seemed normal for those students, PP can't succeed with such conservative way of thinking. If PP is to be applied with undergrads, a more collaborative environment should be encouraged with some other supporting agile practices, which promotes the concepts of "collective ownership of code" and "whole team participation".

6. Considering on-time software delivery over product quality

Although this is the case with most of development teams, it comes into focus with PP teams. One of the major benefits of PP is to ensure software quality because of the instantaneous testing and the ideation that occurs within two minds instead of one. So, if the "quality" is not achieved, PP loses one of its main advantages. The reason for developers, either individuals or in pairs, to sacrifice quality is the limited time. However, pairs in PP sessions should manage to get the best use of available time to produce the required functional product with an acceptable quality level. The problem with the two pairs in our two sessions was with their main goal, which was to "deliver a working piece of software on time", not to "deliver quality software on time". More practices should be put to use to ensure that quality is part of the deliverable, not a complementary feature.

7. Disregarding creative ideas in favor of traditional solutions

Trying new coding approaches, working on unfamiliar tools, and implementing uncommon solutions have always been discouraged with the excuse of "time limits". This fully contradicts with the objectives of PP, where an important one of them is to promote creative solutions and build an innovative environment. Introducing PP to developers should focus on the real objectives behind PP, not just to deal with it as a development technique. The two sessions witnessed some ideas that have been rejected because of the 1-hour time limit, while the objective of PP is to encourage pairs to work on their ideas and try to manage their time to be able to work on their ideas (even by asking for more time if required, as creative solutions are always easy to get approved for more time).

8. Giving less consideration to UI design

The pair on session 1 discussed some aspects related to UI design that led them to assume certain usage scenarios and helped them decide on some interface design issues. However, that part was given a very small amount of time when compared to actual implementation time. On the other hand, session 2 developers didn't consider UI design until the very end of the process, and they didn't discuss user preferences or any usage scenarios. Although discussion and collaboration between pairs would lead to better design decisions (as shown on session 1), UI design should be assigned more time and should be treated carefully by the pairs.

VI. Recommendations for introducing PP to students for collaborative environment

One of the main advantages of collaboration is to promote the culture of agility, where innovation is considered a core value, rather than being a side gain. Based on our discussion above, we are recommending certain practices that will help instructors achieve better results when they introduce PP to their programming classes.

1. For the first PP sessions, students are better not to be paired randomly

It will be better to either let students select their partners, or pair them based on their GPA (or their grades on previous programming classes). This, however, can be changed in later

sessions so that students can be paired with new partners to enhance knowledge transfer among students.

2. Assign the first PP round to high-level design and session planning

Students should be encouraged to spend reasonable amount of time only for software design, UI high-level design, and planning for their development timeline and milestones. Later, students will naturally start working on design and planning without being "required" to do so, as the advantages of spending some time on those non-coding tasks will positively affects their programming work.

3. Introduce quality as a basic requirement, not as a bonus

When presenting the problem statement, ensure that quality should be considered carefully by developers, and include examples on how users will assess the product quality. Quality attributes should address design, code, and UI.

4. Emphasize the role of talking and listening

PP session facilitator should make sure that pairs are in continuous two-sided conversations. Students who are found to be silent for long times should be asked for the reasons and guided to participate in discussions with their partners.

5. Present the assignment as a challenging programming task, not as a regular class assignment

Students get motivated when they feel they are solving a real problem that requires them to be noticeably smart. Working in teams of two makes this feeling even stronger with the higher levels of competition among teams. Therefore, the facilitator should shift students' thinking from "what grade will I get for that task?" to "How good is my task compared to the required level of quality?". This can be done by presenting the culture of agile development, where teams are competing to present the highest quality within the allowed time limits.

6. Students should be asked to explicitly consider user context and usage scenarios

Aside from UI issues, students need to spend enough time discussing who will use their software, how will users deal with it, and what are the non-functional requirements (that may have not been stated on the problem definition). From one side, it gives them more insights about the application they are working on. From another side, that will ensure the software quality as it considers the unclear – yet important – non-functional requirements. Moreover, such brainstorming will get students to come up with innovative ideas that might have not appear if they work based on the given assignment definition on its own.

VII. Conclusions

Integrating usability with agility has been recommended by many researchers and practitioners to achieve various benefits. However, the proposed approaches for such integration haven't deal with many of its problems. Since collaboration among developers is essential for both agile and UX teams, we decided to explore how Pair Programming (PP) can help in creating a better collaborative work environment for developers. We claim that the higher the interaction level among developers using PP, the better UX they will achieve, especially for the mobile app development. Our first step was to investigate how PP can be introduced to developers in small companies or in programming classes, which is the main scope of this paper. We combined our literature review with expert review sessions to come up with some practices to help introducing PP to CS class students, with the objective of helping students collaborate in a disciplined yet flexible way. These practices will be extended to help developers working on real-world development environments.

References

[1] Tichy, W.F. *Agile development: evaluation and experience*. in *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on*. 2004.

[2] Lycett, M., et al., *Migrating agile methods to standardized development practice*. Computer, 2003. **36**(6): p. 79-85.

[3] Highsmith, J. and A. Cockburn, *Agile software development: the business of innovation*. Computer, 2001. **34**(9): p. 120-127.

[4] Sohaib, O. and K. Khan. *Integrating usability engineering and agile software development: A literature review*. in *Computer Design and Applications (ICCDA), 2010 International Conference on*. 2010.

[5] Nielsen, J., *The usability engineering life cycle.* Computer, 1992. **25**(3): p. 12-22.

[6] Chamberlain, S., H. Sharp, and N. Maiden, *Towards a Framework for Integrating Agile Development and User-Centred Design*, in *Extreme Programming and Agile Processes in Software Engineering*, P. Abrahamsson, M. Marchesi, and G. Succi, Editors. 2006, Springer Berlin Heidelberg. p. 143-153.

[7] Ambler, S., *Tailoring Usability into Agile Software Development Projects*, in *Maturing Usability*, E.-C. Law, E. Hvannberg, and G. Cockton, Editors. 2008a, Springer London. p. 75-95.

[8] Lee, J.C., *Embracing agile development of usable software systems*, in *CHI '06 Extended Abstracts on Human Factors in Computing Systems*. 2006, ACM: Montral, Qubec, Canada. p. 1767-1770.

[9] Hodgetts, P. *Experiences integrating sophisticated user experience design practices into agile processes*. in *Agile Conference, 2005. Proceedings*. 2005.

[10] Sy, D., *Adapting Usability Investigations for Agile User-centered Design*. Journal of Usability Studies, 2007. **2**(3): p. 112-130.

[11] Beyer, H., *User-Centered Agile Methods.* Synthesis Lectures on Human-Centered Informatics, 2010. **3**(1): p. 1-71.

[12] Nielsen, J. *Agile Development Projects and Usability*. 2008.

[13] Hussain, Z., W. Slany, and A. Holzinger, *Current State of Agile User-Centered Design: A Survey*, in *Proceedings of the 5th Symposium of the Workgroup Human-Computer Interaction and Usability Engineering of the Austrian Computer Society on HCI and Usability for e-Inclusion*. 2009, Springer-Verlag: Linz, Austria. p. 416-427.

[14] Seyam, M. and G.H. Galal-Edeen, *Traditional versus Agile: The Tragile Framework for Information Systems development.* International Journal of Software Engineering (IJSE), 2011. **4**(1): p. 63-93.

[15] Williams, L. and R. Kessler, *Pair Programming Illuminated.* 2002: Addison-Wesley Longman Publishing Co., Inc. 288.

[16] Beck, K. and C. Andres, *Extreme Programming Explained: Embrace Change*. 2 ed. 2004: Addison-Wesley Professional.

[17] Sutherland, J., *Scrum: The Art of Doing Twice the Work in Half the Time*. 2014: Crown Business.

[18] *Manifesto for Agile Software Development*. 2001 [cited 2015 01/20/2015]; Available from: http://agilemanifesto.org/.

[19] *Principles behind the Agile Manifesto*. 2001 [cited 2015 01/20/2015]; Available from: http://agilemanifesto.org/principles.html.

[20] Williams, L., et al. *Eleven Guidelines for Implementing Pair Programming in the Classroom*. in *Agile, 2008. AGILE '08. Conference.* 2008.