Agile Usability

Kobe Davis
SENG 609.51

Abstract

This paper reviews the concept of agile usability. It provides an overview of the approaches and techniques of interaction design followed by a discussion of the principles of agile development. The remainder of the paper reviews and discusses the current state of union or dis-union between these methodologies via a survey of papers and literature on the topic.

1. Introduction

At first glance and outsider might assume that usability or interaction design would be an implicit part of any software engineering methodology. Unfortunately that assumption would be quite far from the mark. Many software engineering projects seem to be designed with the user interaction and experience as an afterthought.

For example, witness the majority of cell phones on the market, all provide a vast array of tools and added functionality in addition to making and receiving calls. Making use of any of that functionality is often an exercise in frustration, navigating poorly designed menus and confusingly integrated features. There are 'standards' for designing the interaction required to make and receive calls but beyond this engineers have put little thought into the rest of the experience or whether the additional features are even those needed or wanted by the purchasers of the phones.

To address these issues software development needs to involve greater consideration and interaction with the end-users for which the software is being developed. This is a primary goal and practice for a field labeled interaction design. Unfortunately, interaction design is not necessarily a natural extension of software engineering and is often lost in the rush to get a working product out. Agile methods which advocate 'working code' over 'documentation' and the ability to change direction vs. a defined plan may contribute to this tendency to ignore the interactions of the resulting software. The insight needed to determine the real usability of a software product requires some upfront study of the end users of the system and cannot be limited to the requirements of the on-site feedback. The same is true in terms of reviewing the design or finished product. It is not enough to determine if the software 'works' free of bugs, the user experience goes much deeper than this.

The paper reviews the major tenants of interaction design and agile methods. Following this is a review of current literature and a study into integrating the more heavyweight tenants of interaction design with the lightweight process of agile methods. To date agile developers have begun to see the need for more interaction design, but the proponents of interaction design are not nearly as keen to see these practices boiled down to a lightweight iterative approach.

2. Usability - Interaction design

2.1. What is interaction design?

Interaction design, which traces its routes to the study of Human Computer Interaction (HCI), is concerned with improving the design of a product so that the use of the product is efficient, natural and pleasant. The system or product should be intuitive so that it is easy to learn and thereafter the user should be able to easily remember how to use it. It should also strive to provide the utmost utility for the experienced user without compromising the original simplicity. At the end of the day the aim is to provide the best possible experience for the user.

Interaction design extends beyond just software to physical products and other systems and environments. In all cases looking to provide a product or system that is effective for the user and does not necessarily require the user to adapt to it. Although interaction design has a broad scope that continues to grow we will focus on interaction design in the context of developing software. Specifically those principals and practices that apply directly to improving the experience of a software product as it is being developed

As per Sharp et al [1], one can hope to arrive at the best possible user experience by relying on intuition, or they may be rigorous in their approach to understanding the user and making their design choices. They suggest the following points be examined:

- Taking into account what people are good and bad at.
- Considering what might help people with the way they currently do things.

- Thinking through what might provide a quality user experience.
- Listening to what people want and getting them involved in the design.
- Using, 'tried and tested' user-based techniques during the design process.

All of these items point to taking a more in depth approach to determining user requirements than simply writing down the requests of the user and then implementing the code based on the developers current knowledge of the system. Interaction design requires that the designer have a more in depth up-front knowledge of the real user's of the system, including how they currently work and accomplish tasks. (By real users we do not necessarily mean the sponsor of the software or the on-site expert.) The end goal is to [2] "design interactive products to support the way people communicate and interact in their everyday lives". To the software developer this means a very user-centric approach involving investigation and understanding of the systems users as well as communication during and after the design/development process. It also means the simplest solution may not be the most useful one (although in some cases, as per the cell phone example, a simpler solution may be the order of the day).

2.2. Interaction design process

Interaction design is a 'user-centered' process in which the foundation of the process is to involve end-users of the system from the beginning of the design process to the end. The argument is made that without the involvement of end-user it is very difficult to develop a solution that adequately supports their needs. A representative or manager providing requirements or guidance for the system may be able to provide the functional goals but is not necessarily aware of the day to day details of the users work environment and how they currently accomplish their tasks. Often the requirements provided deal with the ideal scenario and ignore the actual flow of the process.

Involving end-users in the design of the system ensures that their day-to-day activities are taken into account. It is also an effective way to manage expectations. Rather than operating on a wait and see basis users are directly involved in providing requirements for the system. From an early stage users know what is possible and what is not and ultimately are able to ensure that the system that will be effective for them when it is delivered. This is opposed to the system or functionality that is essentially dropped in their lap. In this scenario anything that is not necessarily as per expectation is a source for disgruntlement vs. the scenario where a user has been able to provide input and knows what to expect and the reason why the functionality is the way it is.

Users may be involved in the project in varying degrees, from assigning them direct roles in the development of the design of product, to simply providing time to ascertain needs directly and allow for feedback at various intervals. The level of involvement in the users is a source for debate and best left to the decision of those involved in regard to what is appropriate for the system. A short-term web development project may require only limited involvement whereas the long-term development of a critical or fundamental system would necessitate higher user involvement.

Interaction design involves four basic practices as laid out by Sharp et al. [3]:

2.2.1. Needs and Requirements. The first practice involves identifying the end-users of the system and their needs. These needs form the under-pinning of the requirements and allow us to begin shaping the end user-experience. Without understanding the user's needs directly there is risk in designing something that is not as effective as it could be. This input is gathered through study and/or interaction with the users and analysis of the information that is gathered.

2.2.2. Alternative Designs. The second practice involves developing prototype designs based on the requirements. These prototypes are the result of conceptual and physical design. The conceptual design provides the high level parameters of the system regarding the what and how. The physical design provides the lower level details concerning the actual look and layout of the system.

2.2.3. Interactive models. Third on the list of practices is creating an interactive model of the design(s) created in step two. This is a physical (or electronic) implementation of the design that the users may actually analyze and respond to. The model may simply be a mock-up and not necessarily a working implementation, but it is enough to get a sense of what works and what does not.

2.2.4. Evaluation. The final practice involves evaluating the designs and models created in the prior steps. Interaction design is an iterative process that seeks to improve the end product through continuous evaluation. The interactive models are tested to see how well they support the original requirements along with usability criteria such as how easy it is for the user to learn and the software. Do they make mistakes and/or find it frustrating. This feedback is then taken back to create better designs and as a result better models.

It is important to note that these four practices form a lifecycle model of their own, similar to the numerous

lifecycle models that exist in software engineering. Sharp et al. provide the following visualization of the model in Figure 1. You can see that the model forms an iterative or continuous feedback loop and is not necessarily a pure waterfall approach to development. This is similar in many ways to the various agile software development models.

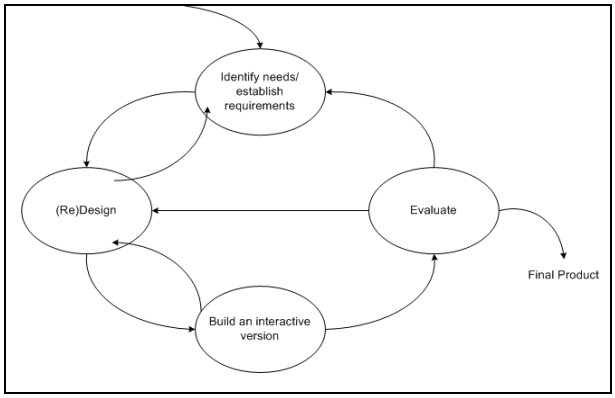

Figure 1 Simple interaction design lifecycle model

In the remainder of this section we will expand on these four practices and provide more detail on the interaction design model as a whole.

2.3. Needs and Requirements

As previously described, it is not enough with interactive design to simply accept a set of requirements without further identifying the end-users of the system and understanding their needs and goals. Only then do the designers have the information required to establish a strong and stable set of requirements. Without this in-depth understanding of the users it is not possible to ensure a set of requirements that are as close to meeting the end goals of the users as possible. Often requirements are developed by users whom have not entirely thought things through in detail or have simply provided high-level needs. Without the understanding of the users and their needs the development team is in no position to refute or provide insight into what is actually required. Establishing the requirements requires first gathering data about the users.

2.3.1. Data gathering is the primary activity conducted to either produce or support the needs and requirements of the users. Data gathering consists of a number of common techniques including interviews, questionnaires and observation as well as less direct methods such as studying documentation or reviewing similar products. A formal or informal set of goals is recommended to start the data gathering process followed by establishing a relationship with the participants, whether this involves establishing consent or simply outlining the goals as it relates to the participants job. The actual data gathering may involve the use of more than one of the techniques mentioned above to allow for corroboration of the information received. It may also be advisable to conduct a trial run to ensure the data gathering techniques proposed are actually viable for the particular situation.

Interviews are the first technique listed for data gathering and are have similar goals to the reporter or TV show host who interviews a subject to allow them the opportunity to express their views and/or insight into the matter. The reporter or host may have a specific 'structured' list of questions they want to ask or may allow for more 'unstructured' dialogue while still guiding the conversation and keeping it relevant to the subject at hand. Even in the scenario where the interviewer has a specific list of questions the questions may be closed requiring a specific confirmation or denial. Alternatively, open-ended questions allow the interviewee a chance to expand on the subject and provide further detail. This type of data gathering provides rich, first-hand knowledge of the subject and needs.

Questionnaires are another technique for gathering information on the users needs and may consist of simple surveys with specific choices for answers to more open-ended questionnaires created to gather opinions. The upfront design of a questionnaire is very important, unlike the interview there is no opportunity to re-phrase or expand on a question. All questions must be clear and concise. The possible answers to questions may take a number of formats ranging from written answers to selections from a set of possibilities or ratings on a scale. The questionnaire may also be administered via paper, electronically or potentially over the phone. A questionnaire may be more effective in reaching larger, broader populations of users who have an incentive to provide feedback without being directly questioned.

The final technique, which directly involves the user, while not necessarily providing direct feedback is observation. Observation is useful in understanding the context in which the users accomplish their tasks. At the outset of the design this provides insight into the goals of the users and in the latter stages can provide feedback with respect to the design itself. Observation is useful in providing insight that the user might not express or be able to express. Examples include pleasure or displeasure with certain aspects of the system that are not necessarily mentioned in interviews or feedback. Also it may be easier to show someone how a process is accomplished versus describing it in detail.

Observation may take place 'in the field', or in other words the user's workplace or the natural setting for the task. Observation may also take place in a more formal,

controlled environment such as a usability laboratory where the user's actions and reactions may be studied in detail. User's may also be indirectly observed through diaries or logging of activities. The diary involves the user recording actions as they are performed whereas logging involves instrumentation of the software to record actions as they occur. The different types of observation may be used on their own or in conjunction as the situation requires.

2.3.2. Data analysis is the second step in determining needs and requirements. The types of analysis performed are in part determined by the goals established at the outset of the data gathering and the actual types of data gathering performed. Data analysis falls under two broad categorizations including quantitative and qualitative analysis. Quantitative data and analysis involves information that is numerically based and is used to determine a concrete number or average that relates to the item(s) being studied. If you can use statistics or excel chart to analyze and represent the data it is quantitative. Qualitative data is less concrete and is not as easy to measure. It provides more abstract observation or description and therefore qualitative analysis helps us provide the relative sense of something vs. concrete values.

There are a number of tools and/or theories available to help in the analysis of the data. Whether the designer is involved in qualitative or quantitative analysis the goal is to categorize and summarize the observations so that they may draw logical conclusions as to the needs and goals of the users. The designer should be able to present the analysis in a format that supports these conclusions. This may involve rigorous notation such as UML or less detailed output such as a simple story summarizing the results.

2.3.2. Establishing requirements is the conclusion to the data gathering and analysis efforts. In this context the term requirements encompasses both functional and non-functional requirements, which describe what the system should do and the constraints around its operation. The requirements can be further broken down into functional requirements, data requirements, contextual or environmental requirements, user characteristics and usability or user experience goals. These categorizations of requirements ensure that the full range of possibilities is considered and results in a more stable outcome at the end of the cycle

Requirements can be documented in a number of ways including writing task descriptions that may be broken down into scenarios, use cases and essential use cases. Scenarios are descriptions of the tasks to be performed written in the users language. The primary focus is the users goals and reference to technology should not be made. Use cases take the scenario one step further by focusing on the interaction between the users and the system. Use cases are one and the same with the use cases found in object-oriented modeling and describe the interactions a user or 'actor' has with a system to accomplish a goal. Finally, essential use cases are meant to provide a third point of view, one that provides a more abstract view than a scenario and does not assume the interaction with the system that a use case does

2.4. Alternative Designs

Once a concrete set of requirements have been established based on an understanding of who the end users of the system are and their needs and goals one can set about designing a solution to meet those criteria. Given that the conceptual design arises as a direct result of the established requirements interaction design requires that the designer have a solid understanding of those requirements along with empathy for the user's goals. With this understanding, ideas for the user experience will begin to arise and the validity can then be discussed with all stakeholders..

The conceptual design will consider a number of aspects relating to the implementation of the user interface. The first of these aspects concerns the metaphors that are appropriate for the system. A business application may have less flexibility in this area but systems designed for other uses need not be limited to forms and checkboxes. For example, an application for purchasing movie tickets may be modeled on an actual ticket counter and provide options in the same way as if a user was standing at a physical counter.

This leads into the types of interactions that the design should entail. Possible types of interaction include instructing, conversing, manipulating and exploring. A typical system will include multiple types of interaction. Instructing entails users entering commands and selecting options from menus as with a typical business oriented application. Conversing may be more typical of a search engine where a user enters a question or search term and the application responds. Manipulating may be more familiar with a design environment such as drawing or drafting application where users interact with a virtual model. Finally, exploring is more typical of the many game environments of today and virtual reality worlds promised of tomorrow.

The final step in establishing the conceptual design is deciding on the type of interface that is required for the system. This is based on the interactions the users will have with the system. Even in the more limited world of HCI there are still many choices or paradigms to evaluate such as a web browser or client based platform on a full sized screen or possibly a mobile or tablet based

application on a much smaller device offering a different set of controls. Only once the designer has determined the type of interface that is appropriate and the interactions available based on those choices can they set about creating designs.

When constructing the initial conceptual design the designer considers all of these issues in light of the functions the system needs to perform and how they relate to each other. This information again comes from the requirements and understanding that have been previously established. Along with the functions the system performs the design also needs to consider the data or information that the system needs to make available to the user in order to perform the functions. Collectively these ideas give the conceptual design a head start in providing the best user experience possible.

2.5. Interactive Models

Creation of the conceptual design leads directly into the physical creation of a conceptual model or prototype. It is not necessarily a requirement that the conceptual design be complete before the model is constructed as the model can be used as input and feedback into improving the original design. The designer should start with a 'low-fidelity' approach to the first models, which involve pen and paper based storyboards created from the scenarios described in the requirements gathering. The creating of the storyboard provides a picture representation of the scenario for easy feedback and also forces further thought and analysis with respect to the scenario.

The next step in the low-fidelity models is to create card based prototypes from use-cases. Using the storyboards created along with the use-case descriptions they can quickly model user interfaces on cards with basic user inputs. The card-based approach may then be shard with users of the system to garner feedback on the effectiveness of the implementation and any limitations of the interface.

The card-based approach then evolves into more detailed sketches of the user interface on paper or into electronic interfaces mocked up via a GUI design tool (without actually implementing any code). This model provides a paper-based representation of the user interface that a potential user may interact with as if it were the real thing. A team of individuals present the user with the interface mockups and provide the correct 'screens' while the user simulates use of the interface while also observing the actions of the user and recording the missteps and problem areas.

This low fidelity approach to prototyping allows us to quickly evaluate potential interfaces and provides an initial look at the user experience as a whole. The designer is able to determine what works and detect problem areas and gaps in the implementation without creating costly physical implementations. The designer or team may iterate through a number of paper-based implementations before moving on to creating more high-fidelity implementations that may mirror or serve as a starting point for the implementation of the final interface. The aim is to explore as many options while gathering continuous feedback before starting on the concrete implementation.

2.5. Evaluation

The final practice in the process is evaluation. Although it is listed as the last stage of the process , evaluation may at occur at any stage in the product development life cycle. It may be performed on early prototypes to ensure their validity and to provide additional feedback in the evolution towards a final design. It may also be performed on a finished product to refine or evolve the usability of that product or validate that nothing has been 'lost in translation'.

At this stage the designer performs quality assurance and garners feedback on the product or system to ensure that it is usable by the end-user. Going beyond just being usable, the designer looks to see that the product provides an enjoyable experience and effectively meets the goals of the user. There are many aspects of the system or model that can be broken down and evaluated on an individual basis. It is not necessarily effective to garner feedback on the whole model at once. Similarly there are different scenarios or locations for evaluating the model. Internal and/or more detailed checks may be performed in the lab while user reaction to usefulness and experience is better evaluated in a natural setting.

At the highest level evaluation is broken down into three approaches: usability testing, field studies and analytical evaluation. Usability testing is the oldest of the evaluation approaches having been around for many years. It involves providing the user with a model of the system, or system itself and a task to perform. The user is observed while accomplishing the task and the time it takes to complete it and any errors made are recorded. The user may also be interviewed after the fact for further feedback.

There are a number of user testing methods that may be performed as part of usability testing. At the core of each of these methods is quantitative analysis of the user performing the task. This usually takes place in some form of controlled environment free from distraction and competing interests. The output of usability testing can be used to form standards for acceptance that can then be used in later iterations or revisions.

Field studies, as opposed to usability testing are performed in the natural setting for the use of the product

or system. They relate strongly to the observation techniques employed to gather data on the users and their needs and can be used for a variety of reasons. They can be employed at the outset of a project to determine usages of the current system and opportunities for evolution as well as for evaluation of new products/features and easing their introduction.

The final evaluation approach, analytical evaluations, involves a more rigorous approach to evaluation of the system. This approach does not involve the end users directly through observation, questioning or testing. Rather, inspections are performed by expert users to identify common issues based on walkthroughs or heuristics. Alternatively, theoretical models are employed to compare the system with expected optimal performance models.

Designers may choose to employee a single approach or multiple approaches in conjunction. There are no rules regarding their deployment. They may also be performed as part of a detailed testing phase or on ad-hoc basis at any point in the development lifecycle.

2.5. Interaction Design in Summary

We can see that interaction design is a thorough approach to understanding the end-user and their needs. Along the way we iteratively develop extensive models and prototypes to evaluate and decipher the best approaches to providing a pleasing user experience. The individual practices are hard to argue with and seemingly lend themselves to an agile approach. The main source of debate is the traditional or up-front approach that true interaction design requires when contrasted with the 'start immediately' approach of agile methodologies. Interaction design also argues for a core set of specialists dedicated to these tasks and providing the communication channel between users and development.

The next section provides a short review of agile methodologies. This is followed by a review of experiences and observation regarding ideas and actual case studies regarding the integration of the two practices. At the end we hope to gain a better understanding of whether or not it is feasible to employ the usability practices of interaction design on the fly in an integrated team as opposed to up-front interface design done by a stand-alone team.

3. Agile Methodologies

Agile software development methodologies are best described in relation to the traditional, waterfall software development methodologies they were designed to replace. Traditional approaches to software development prescribe a very methodical, design up-front, single path approach to software development. Agile development counters that systems, especially large systems, cannot be completely visualized and documented from the outset. In the agile world change is inevitable and something to be welcomed and planned for as opposed to being avoided at all costs.

The core concepts of any agile development methodology can be best summed up by the tenants of the agile manifesto [4], created in 2001 by a group who dubbed themselves the agile alliance. These individuals were the principal thinkers and orators behind the lightweight software development practices that had begun to be formed in the late 1990's. The tenants of the agile manifesto are as follows:

- Individuals and interactions over process and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Agile software development methodologies place user consultation and development of working code ahead of heavy weight abstract design and abstraction. Development begins based on the first concrete requirements almost immediately and with possibly only a wire frame model of the bigger system outlined. It is expected that there will be multiple iterations and that within each iteration user feedback and involvement will be sought.

A number of different agile software development methodologies have been created, all with the same core tenants and different implementations or practices. These include eXtreme Programming, SCRUM, Feature Driven Development, DSDM and the Rational Unified Process to name a few. The key features of XP and SCRUM are provided here and used as a basis for later discussions in the context of agile usability.

3.1. Extreme Programming

Extreme programming (XP) is based on principles of simplicity, and communication [5]. The rest of the core practices extend from these two principles that also directly relate to the main ideas in the agile manifesto.

XP starts with involving the entire team in the planning and development of the project from beginning to end. Some team members may be more experienced than others or provide different knowledge bases, but all of the team members are present at once. Knowledge is not gathered by a core group and disseminated. Also present and part of the team is the customer, or at the very least a business representative who can speak on behalf of the customer.

XP is an iterative process that starts with a practice called the 'planning game'. Here requirements are

presented in the form of user stories by the customer and the team provides estimates for the time/effort required to develop these stories. The customer then has the ability to sort and plan the priorities for development. In iterations of a several weeks the developers implement the prioritized stories, tracking their velocity and estimates so that the planning process is continually improved upon.

Each of the iterations is considered a small release of the software. At the end of the release each feature that has been developed must pass customer acceptance tests provided as part of the initial requirements. Ideally these tests are implemented prior to implementing the code and are run on a continuous basis as the code is integrated. In this way the testing guides and drives the actual development.

Extensive up-front planning and documentation is eschewed in favor of simple designs and avoiding planning too far ahead. Code is designed and implemented to be self-documenting. Programmers also work in pairs and do not own a single piece of the system. In this way all code is developed under constant peer review and developers become intimate with all parts of the system.

3.1. SCRUM

Scrum is another popular agile software development methodology that is concerned largely with the project planning and communication aspects of the development lifecycle. Similar to XP and other agile development process, Scrum seeks to minimize the preparation and lead time it takes to begin 'creating value'. The following image in Figure 2 [6] provides a high-level overview of the Scrum life cycle.

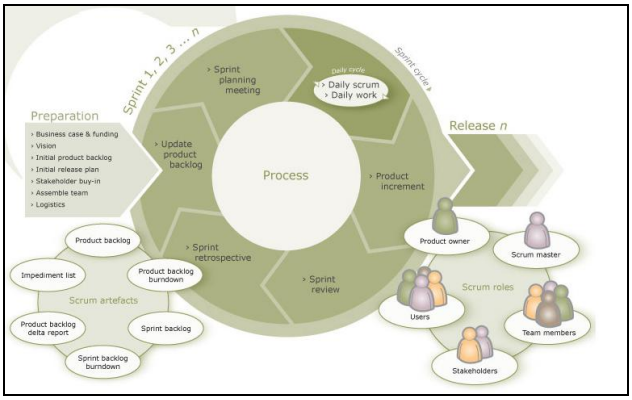

Figure 2 SCRUM Lifecycle

Scrum focuses on the definition/creation of a few roles along with the process of iterating through successive sprints. The more detailed guidance on testing, documentation, design etc. that is found in XP, is left out. Scrum defines three primary roles, product owner, Scrum master and the development team members. The product owner fills a similar spot as that of the customer in XP. While not necessarily being the customer they are a representative of the customer and are responsible for gaining a high level understanding of the project 'backlog' of items in the development queue and prioritizing them accordingly. They accept or reject the completed work and ensure communication between stakeholders while scheduling releases etc.

The SCRUM master may or may not also be called the project manager. They are responsible for guiding the SCRUM process and leading daily scrum meetings and sprint scheduling. They are the backstop for the process and ensure that things keep moving forward with proper communication.

The team members constitute the actual development team including all of the required roles. The team may include analysts, developer, quality assurance, etc.. The key differentiator is that any team member is required to be involved from the beginning of an iteration to the end so that they are 'committed' to its success. There should be no interlopers.

The core of the Scrum approach is very similar in its approaches to iterations, called sprints, and the planning process that calls for collaboration between team members and customers to decide on the goals for the sprint. The most visible (and most often adapted) part of the sprints is the daily Scrum or meeting for all members. All team members attend and provide what has been accomplished since the last meeting, what will be accomplished before the next along with describing any impediments or problems causing delays and hold-up. As a result the entire team is informed of progress and has the ability to provide feedback or insight with regards to impediments.

Similar to XP, at the end of the sprint there is a sprint review and retrospective. These meetings allow for introspection into the recently completed iteration to look at what has gone wrong and right and allow for suggestions for improvement. All aspects of the sprint are up for debate from planning down to design, coding and testing.

4. Bringing Agile and Interaction Design Together

The previous sections have provided an overview of interaction design and its high-level goals, practices and processes. The ideas behind agile development methodologies have also been reviewed along with the key points from two of the most popular agile methodologies. At this point we will try to explore the state of union, or dis-union as the case may be, between these complementary but disjoint camps. A review of papers and texts which cover this topic has been

conducted and despite the fact there are no definitive answers to be had, we can draw our own thoughts and conclusions regarding best fits are and where things may be headed in this respect.

Most of the literature to be found with regards to the combining of interaction design and agile methods comes from the agile side of the equation. Many individuals or groups who use or advocate agile methods have noticed a gap in this respect and are in favor of introducing more usability design and analysis in the process. Of course the intention is to do this in the most lightweight method possible, without introducing heavyweight design and/or processes

Agile usability does not curry a lot of favor with the usability or interaction design side of the crowd. The views of the authors of the interaction design text and others who are seen as usability experts do not support this notion. The first piece of literature reviewed in this respect is a section from the interaction design text which does afford mention of agile methods more than once.

4.1. Interaction Design

[7] The Interaction Design text presents a case study on a Toronto based company Alias that produces graphical design applications. Alias had already developed extensive customer input methods relating to usability for product development. With plans to develop a new version of their product SketchBook Pro the team wanted to move to a more agile method of development. This required them to integrate those customer input methods into the new agile development process and lifecycle.

The development team settled on a hybrid approach to agile development that made use of aspects of ASD, Scrum and XP. The biggest problem this presented was that the new shortened iterations and timelines presented a challenge for the existing practices of contextual inquiry, interview, usability tests, surveys and beta tests which the interaction designers conducted to identify the target market and collect customer input. These practices had previously preceded the development efforts allowing for a time margin to plan and incorporate the feedback. With the immediate development efforts and short schedules prescribed by agile practices the usability efforts were now squeezed and would possibly be less effective.

The answer to this problem was to create parallel processes or tracks in each of the iterations. In one track the developers worked in a normal iterative cycle implementing sets of features starting immediately at the beginning of the iteration. In the second track the interaction designers conducted interviews and created prototypes for features to be implemented in subsequent releases. They also performed usability tests and gained feedback on features that had been implemented in previous releases. The interaction designers felt there were a number of benefits to this. The agile aspects of having immediate feedback and communication from all sides along with the ability to directly affect the planning process for further iterations were seen as highly beneficial.

This case study provides evidence of a successful combination of the two practices with desirable results. The interaction design team is able to realize the benefits of the more agile lifecycle, which provides constant feedback and the opportunity for and acceptance of change. By the same token the software engineering team realized the benefits of up-front usability and design input when implementing features along with feedback on features that have been implemented in the previous release. Although this orientation somewhat expands the process to require two or three iterations for a full-cycle it is still very agile in terms of its immediate development of features and resistance to a pure waterfall or ‘big up front design’ approach.

The authors of the interaction design text provide a caveat to this case study in a section entitled “Dilemma: How agile should user-centered design become”. The title gives a fairly good indicator of how these interaction design proponents feel about the validity of adapting the usability process to a more agile approach. The main point of contention is that experienced UCD practitioners see an extended period of user research including field studies prior to development work as mandatory. The authors pose the question as to how much UCD work and time should be spent before development work should begin. This seems to be a very defensive stance and one similar to the waterfall approach that agile methods were designed to counter. The short ‘agile’ answer is of course that as little time as possible needs to be spent up-front so that development can begin and value can be created. The case study showed the non-UCD practices can begin in tandem with the UCD work which then provides input and feedback at later stages.

4.2. XP vs. Interaction Design

[8] In an interview conducted with Kent Beck and Alan Cooper we again find similar attitudes as found in reading the Interaction Design text. On the part of the agile methodologies proponent we find an openness to adapt and integrate aspects of UCD into agile practices. Or maybe it is a matter of bringing agile practices to UCD? On the other hand we find that the proponent of UCD is very skeptical and unwilling to accept the idea of modifying the practices prescribed by UCD. Kent Beck is the proponent, and one of the originators, of the XP process along with Agile Methodologies. Alan Cooper is a strong UCD proponent whom has often written and spoken to that end.

In the interview, Kent Beck makes an argument for a process similar to that found in the case study of Alias and their development of SketchBook Pro. In that situation the usability practices have been integrated into an agile framework. Allan Cooper makes the counterpoint, with arguments similar to that of the authors of the Interaction Design text, that interaction design is an up-front process that needs to be completed in full prior to development beginning. Cooper also makes the statement that once development or coding begins, a trajectory is set that cannot be modified later on, yet this is the specifically one of the areas that agile methodologies attempts to address.

Allan Cooper argues extensively for the creation of a dedicated interaction design team that is an agent for organizational change and exists between the customer or stakeholder and the development team. It becomes their responsibility to coordinate requirements and input with the results of creating a detailed plan for the development. This is similar he argues, to the architect who creates the models and drawings for a building with input of the customer, while also consulting the engineer to determine what is and is not possible. This does not coincide with the view of Beck and others, that software design and development is not like the architecture and construction of buildings. The construction of physical buildings while leaving a fair mount of room for imagination is a fairly standardized problem set which follows strict rules and regulations with established practices for construction. The rules and practices are evolving, but at nowhere near the rate of software development.

Software development is not based on physical medium and therefore the possibilities and problem sets are much larger. Although the practices and processes underlying software development are evolving and growing they are not nearly as established as something like the construction industry, which has a considerably longer history. More importantly the possibilities with software design and development are expanding extremely fast. For these reasons the consideration of usability and interaction are important but by the same token, it is difficult to assume that all possibilities can be planned for without having a basis and starting point to work from.

The dialogue between Kent Beck and Allan Cooper ends in a somewhat of a stalemate. Kent is willing to accept user interaction design principles into the scope of agile development but by the same token, Allan is unwilling to see the possibility of using interaction design without a much more focused team and upfront design effort.

4.3. Are Agile Methods Good For Design?

[9] This is again the same question posed by Preece et al in the Interaction Design text but is put forth by John Armitage in his paper of the same name. We already have an idea of what type of answer we would receive from Preece et all., along with a definitive answer from Allan Cooper. John is a software designer and by the same toke skeptical of agile methods. That said he has managed to find some common ground for accepting or allowing design practices to be integrated in to agile methods.

John repeats the important point that has been alluded to previously, "the agile community rarely mentions users or user interfaces at all, which means that either they neglect the user experience or are focusing on projects with less need for sophistication in user experience (UE)." Through personal experience it follows that UCD is easily ignored, hence the need for integration of usability and agility. It is apparent that John's preference, in a perfect world, would be more and stronger software designers and architects who could provide this vision. He also concedes that, "software's mutable nature (compared to other product types) makes its development more susceptible to (or accommodating of) change. Software has no raw materials or physical structure to scrap, has fewer standards for quality, has low manufacturing and tooling costs, and has very rapid rates of innovation and evolution." This means that working in an iterative environment where change is expected is more likely the reality.

John's answer, as depicted in **Error! Reference source not found.** is to split time between design and implementation on an iterative basis. He even goes as far to as to say that agile methods can improve the design process with the iterative releases serving as ongoing usability tests.

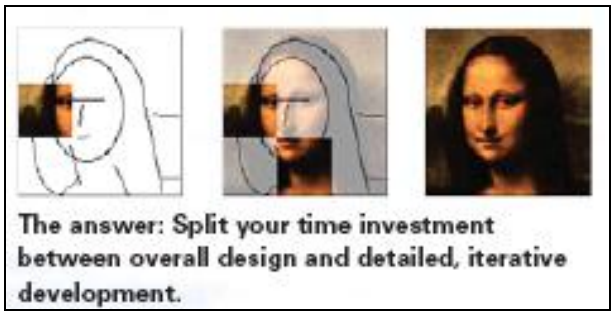

Figure 3

In the paper John also provides example of integrating design practices into the practices of agile projects with which he was involved. The approached describe was to attack design efforts on multiple levels in parallel, the first level involved providing input to the iterations, the second allowed for redesigns of interfaces in existing iterations, while the third provided and over-arching high-level design. This approach, depicted in Figure 4 is very similar to the parallel processes described by the Alias case study and provides further weight to the notion that

UCD techniques can be adapted to a more iterative environment with success not completely dependant on separate, up front design.

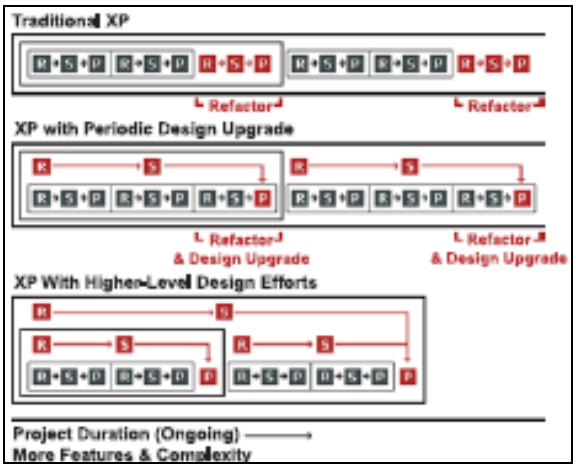

Figure 4

4.4. Agile Usage-Centered Design

[10] Larry Constantine's paper on agile usage-centered design also promotes the idea that agile usability is a viable topic. Although he provides caveats that agile methodologies are not a silver bullet, especially on very large or complex projects, they do have their place within shorter-term projects and smaller teams. Constantine's work has focused around usage-centered design and the philosophies, tools and processes that allow individuals to create designs as quickly as possible. In that sense an affinity has been found between agile processes and usage-centered design.

[11] Scott Ambler also makes a similar case in his paper covering experience activities on agile development projects. Scott makes reference to the interview between Beck and Cooper and suggests there is likely more common ground to be found than is suggested though the eyes of two individuals at the extremes. Scott's primary reference point is Agile Model Driven Development, the agile approach to more traditional approaches such as the Object Management Group's Model Driven Architecture.

Constantine describes a number of processes and techniques to enable and improve upon usage centered design. These tie directly to both the core practices described by interaction design text as well as the agile design techniques touched on by Jeff Patton whose work is discussed in a later section. At the heart of Constantine's recommendation is the use of modeling and prototypes along with constant communication with real users. Again these are topics laid out by the interaction design process. Constantine says that in the lightweight case, modeling need only be based on cards that provide visual and communicative references. The recommendation again, as with John Armitage, is that business logic can be coded independently and in parallel with the design of the user interfaces. The interfaces are then fleshed out in subsequent iterations.

Constantine does contend that a minimal amount of big design up front is needed to create a wire frame map or model of the user interface architecture to guide later development. This is actually not in contention with agile development which also proceeds best with a lightweight overarching direction as opposed to starting implementation with no sense of the bigger picture.

Ambler's focuses on AMDD calls for lightweight modeling preceding each round of development, rather than modeling proceeding in parallel. This is similar to interaction design, which calls for extensive modeling, and re-modeling based on input to extract a final design. Although Ambler's is a much lighter approach which simply calls for some form of modeling and up-front thinking to precede the actual writing of code. This approach does not prescribe or discuss the need for a separate set of roles dedicated to this process. The same team or individuals are responsible for creating the models as those creating the code. Nor does it discuss the need for interaction with end users or outside feedback.

4.5. Agile Usability in Practice

The following section reviews several papers that present further case studies and insight into the actual use of usability or interaction design practices within an agile development environment. The first [12] and second [13] papers present reviews of different individuals and teams with UCD backgrounds and their experiences and views after being involved with agile development groups. These experiences come from those who were both internal (employees) to the organizations as well as working externally (contractors). The third paper considered [14], focuses on the use of a single usability technique, 'usability testing' within an agile development environment. Although this is only part of the truth as the team in this case does make use of interaction techniques such as prototyping and interactive modeling as described by interaction design to support the usability testing.

The first paper from McInerny and Mauer presents reviews of the use of UCD techniques in three different projects and scenarios. Two of these projects are at organizations with previously established UCD group who also make use of agile development environments. The third scenario involves a startup company using agile development, which has hired an outside consultant to provide UCD experience. In only one scenario was up front user research conducted. In this case 'contextual inquiry and personas' were completed prior to the agile development work beginning. In large part, the effort of

the UCD experts was dedicated to providing internal design or consultation during releases.

Common to all three case studies or individuals is the theme of providing low-fidelity design options as well as establishing a high-level guideline for GUI development. Specifically, each individual provides multiple prototypes or design options within an iteration or as input to a subsequent iteration. This is seen as beneficial, as opposed to the single option that the software developers tend to create and run with. There are no conclusions or guidance as to the absolute or perceived benefit of including these practices within the iterations but we can determine that it is possible to work and provide benefit within this scenario as opposed to 'big design up front'. This is also similar to the approach prescribed by Scott Ambler using agile model driven development.

One of the three groups has demonstrated success in implementing usability testing at the end of or within releases. The feedback from this testing is then used to update software developed within the current release. Unfortunately this contrasts with the reports from the other two teams who have not achieved near the same success, in one instance the testing only provides guidance to future planning as opposed to actual change. In the second instance the pace of agile development is found to be difficult for usability testing and as result is conducted only on an ad-hoc basis.

The second paper presents the feedback from a team of UCD experts who were invited to join an agile project already underway. The case study covers an eight-week period of development during which a functional prototype is successfully developed for a worldwide conference. In this study, although it was requested that the UCD group be immersed within the agile group to adopt the same culture, this did not occur immediately. It was decided, from the standpoint of providing a holistic design viewpoint and preserving distinctions in working styles that they remain separate initially with dedicated lines of communication

As with the previous case studies the design group's primary focus was to provide alternate design options for consideration as early as possible. As the timelines were much shorter than the UCD specialists were accustomed to, the designs were created as black and white wire frames as opposed to the high-fidelity versions they would normally have created. When presented, these designs had the effect of provoking questions about the direction of the user interface as well as providing a visual reference for further discussion. The provision of different visions early on in the project established the team as playing a key role. The UCD team in this case was able to provide a high level, conceptual view of the product via the interface that had previously not been considered. This view could then be referenced and updated with each successive iteration.

This case study made a number of strong conclusions in favor of agile methods from the UCD point of view. First, that the iterative development methodology fits well with the cyclical approaches to design employed by UCD specialists, i.e. models and prototypes are created and then updated based on feedback. Second, that designs and prototypes are able to provide a holistic high-level view of the software from the user-interface point of view, which could then be referenced and modified. Agile methods do not necessarily preclude high-level up-front planning but as we've discussed any up front planning usually comes at the expense of thought around the user interface. This ties to the second and third points made that the inclusion of UCD means that agility and simplicity did not come at the expense of a quality user experience. Finally, the authors pointed out that the inclusion of UCD provides a strong backstop to the agile goal of user inclusion and feedback. With UCD, the primary focus is understanding the user, this is only one of the goals of agile development that may oft be ignored or forgotten, the UCD aspect re-enforces this.

The third paper reviews the success of introducing the interaction design or UCD process of usability testing into an agile development environment. The individuals in this case are not usability specialists but have learned of the practices through Jeff Patton's tutorial [15] on the subject. The title and subject of this case study centers on usability testing but in fact the use of interaction design prescribed techniques of prototypes and interactive modeling are also employed to support this goal.

In this cases study the team is constructing a web-based application over a number of iterations. As with the case described in a previous paper, the first iteration is completed on a pure agile development basis with no usability input. At the end of the first iteration, during the retrospective it is decided to employ prototyping and interactive modeling to try and circumvent the usability issues discovered as a result of the lack of planning on this aspect in the first iteration. The use of modeling also means the inclusion of more domain knowledge or user consultation than was present in the first iteration. Of course, these are all items discussed up front by interaction design.

In the second iteration the team created paper prototypes of the user interfaces based on screenshots from work in the first iteration. These were simple artifacts with buttons and controls literally 'cut and pasted' on. Domain experts were then asked to participate in a usability session with the promise of being involved in improving the user experience that was created with disappointing results in the first iteration. With the users cooperation three interactive modeling sessions then took place with the team members taking on the role of the computer or application including a user standing in as the help system. The sessions provided numerous

observations and feedback for the current designs. The end result was fairly significant changes in priorities and assumptions regarding certain features. Features that were previously thought out of scope were added to improve the experience and other less valuable features were dropped.

The paper prototype developed for the usability testing was also posted in a common space along with the comments and feedback that had been gathered. This 'storyboard' created a guide and central point of planning for the next iteration. User stories were tacked onto the storyboard and filled out as the team worked through the user interface. When enough information had been gathered the story was then placed into the queue for possible development. The storyboard was also used as a focal point for further end-user interaction when business process changes were discussed with stakeholders.

A second round of usability testing was completed after the core usability stories from the first round were completed. This time the usability testing was planned for and budgeted up front. The testing was conducted on the application using it as the model/prototype instead of the paper prototype previously employed.

In the end the team found the interaction design or usability techniques benefited them in a number of similar areas as we have found in the previous case studies. The first is that the development of a prototype interface provided a central vision or the whole team, one that promoted and allowed for further thinking and discussion. The prototype also provided the same vision for end users and increased their acceptance and 'buy-in'. Finally, the use of usability testing and prototyping did not hinder but actually helped in the iteration planning.

5. An Agile Usability Methodology?

5.1. Foundations for Agile Usability

In his paper, 'Hitting the Target: Adding Interaction Design to Agile Software Development", Jeff Patton discusses many of the same sources found in this paper en-route to describing his own brand of techniques and practices for adapting UCD/interaction design to agile development techniques. Jeff's techniques are in fact developed based on the UCD techniques described by Constantine and are also the background for the usability testing and prototyping performed by the team in previous section. Jeff has gone on from this original paper to develop a tutorial session and a book (as yet un-finished), which continue to flesh out these ideas and practices in further detail.

Jeff's background, as with numerous others, includes development using heavyweight waterfall methodologies and/or complete chaos followed by a conversion to agile development methodologies, specifically XP. The use of agile methodologies greatly improved the development practices improving the accuracy of scope and timelines. Their use still left gaps though, and as he puts it "…I still found the company missing targets. The resulting product seemed to have features the actual end user didn't need or care about while lacking features the end user did need. Much time seemed to be spent on features that would go un-used by actual end users.'

After attending a week long course provided by Larry Constantine and Lucy Lockwood on user-centered design, Jeff saw potential in adapting these design principles in a more agile way. (At the time Constantine's work was still very traditional in its approach) Jeff's techniques still capture the main techniques of user identification and understanding along with the development of visual models and prototypes to be used for further interaction. Specifically his techniques center on identifying the end-users and their goals and then identifying these roles and actions on CRC cards in interactive sessions. The interactive session allows the team members to start from a clear playing field and develop a common model and understanding centered on the user's point of view. The agile basis for this is the use of only CRC cards for documentation and modeling as opposed to more high-fidelity approaches. This low-fidelity model is later updated to create wire frame models of the user interface.

Jeff found the group session to be an effective way for the team to learn and interact together. Especially in light of the fact that the session started with a chance to vent and leave any pre-conceptions behind. Again, the low fidelity model provided a way for everyone to conceptualize their ideas together and re-work or move things around. There were drawbacks however that sessions were long and tiring, in the same way that the constant collaboration of pair programming is subtly draining. Also some of the concepts and definitions were not necessarily intuitive and took time for the team members to grasp and come around to.

Ultimately Jeff has found these techniques to be fruitful and a natural extension of the agile development methodologies. As he put it, as he would not consider writing code without designing tests first, he would no longer consider designing or developing a solution without first understanding the end-users and their goals.

5.1. Agile Requirements

As previously mentioned Jeff is now in the process of completing a book [16] which continues to flesh out these practices. Although in-complete and as yet un-titled, the book discusses many of the concepts touched on by the interaction design crowd. These ideas are reviewed in a practical and agile-focused context. Unlike the interaction design reference, Jeff does not distinguish between

interaction designers and software developers. He simply provides best practices for the requirements gathering side of the equation and does not require it to be a BUFD process.

The main focus, as with interaction design, is interaction or talking with users and understanding the context of their requirements. A key focal point for facilitating this collaboration design is the use of modeling in various contexts. Jeff reviews practices such as interviews and observation of users, interactive modeling sessions creating low fidelity models with CRC cards, user interface inspections and usability testing. The majority of these techniques are well established, but have been presented again in a lighter weight approach that seems a bit more practical than theoretical.

6. Summary

We have reviewed the core principles and practices of interaction design as well as those driving the array of agile development methodologies. Even on their own we can see merits of including the user-centered techniques within the agile approach which although it is centered on the principles of communication and user-interaction. It does so on an overall project plan and functionality basis and loses sit of the actual user experience of the software.

A number of articles have been reviewed which provide evidence for (or against) this idea. Most in fact provide support and evidence for the benefits of including user centered design practices within an agile framework. Although staunch interaction design and usability practitioners will argue for up-front design and separate teams we can see there are obvious benefits from a hybrid approach. We have seen over and over the following recurring themes, which are described as principles and practices of interaction design:

- Design or prototypes created prior to development either within the current iteration or in a previous iteration (parallel streams).
- The use of lightweight techniques for creating user interface models and prototypes, paper and/or wire frame.
- The creation of multiple options and designs.
- The creation of models allows for a central vision and working point for development and stakeholders (users).
- Usability testing of prototypes provides valuable feedback and buy-in.

We can see that the interaction design techniques and practices do actually fit well within the agile development methodologies. In fact more than one team has used various aspects to improve the outcomes of the work. This is also an under current of Jeff Patton's book which seeks to integrate interaction design with an iterative process.

Working in an iterative/agile software development environment I can also attest to the tendency to gather initial requirements from the end stakeholders but then ignore the notion of further consulting actual user's until the code has been written. This leads to software that meets the requirements and is intuitive from the development team's point of view, but does not provide a pleasant experience for the actual user.

A large piece of functionality was recently completed by a senior developer who, for bettor or worse was allowed to complete the research and develop with very little consultation of other individuals or users. The functionality, now completed solves a very complex problem set, but at the same time is extremely un-intuitive to use requiring multiple training sessions with users to enable them to grasp its setup and flow. Terminology and names for new concepts by the same token is derived from a programmer's lexicon and is not the terminology that and end user would have picked or is comfortable with.

Usability feedback has subsequently been provided, but at this point only minor changes can be made without upsetting the entire apple cart. Had the use of some of the techniques and practices previously presented such as multiple designs and/or early prototypes that are subjected to user review, the end result would likely have been a much better user experience. This did not require complete up-front design as the problem set was know, it was the execution that was lacking and could have been reviewed over one or more iterations.

Enhanced usability practices are something we will seek to integrate within the current development practices with the aim of providing a product that consistently provides an excellent user experience. It will likely start small with increased user consultation and hopefully evolve to include more modeling and prototyping with the requisite reviews.

7. References

[1][2][3][4][7] Sharp, Rogers, Preece, Interaction Design 2nd Edition, Wiley, England, 2007

[5] Xprogramming.com: An Extreme Resource. Available at www.xprogramming.com/

[6] http://scrumforteamsystem.com/

[8] Nelson, E. Extreme Programming vs. Interaction Design. FTP Online. Available at www.fawcette.com/

[9] John Armitage, *Are Agile Methods Good For Design?,* Interactions, Volume 11, Issue 1, pgs 14-23

[10] Constantine *Process Agility and Software Usability: Toward Lightweight Usage-Centered Design*, Available at http://www.foruse.com/

[11] Scott Ambler, *Introduction to Agile Usability: User Experience Activities on Agile Development Projects*, Available at http://www.agilemodeling.com/

[12] McInerney, Mauer, *UCD in Agile Projects: Dream Team or Odd Couple?*, Interactions Nov/Dec 2005

[13] Lievesley, Lee, *The Role of the Interaction Design in an Agile Software Development Process,* CHI 2006

[14] Meszaros, Aston, *Adding Usability Testing to and Agile Project, i*

[15] Jeff Patton, *'Hitting the Target: Adding Interaction Design to Agile Software Development'*

[16] Jeff Patton, *new book…untitled, un-published*

Additional sources

- David Anderson, 'Agile Management for Software Engineering', Prentice, 2003
- Beck, Fowler, 'Planning Extreme Programming', Addison-Wesley, 2000