Volume 5, Issue 9, September 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

# Revising Res PCT to an Open-Source Software Lifecycle

**Adnan Shaout, Cassandra L. Ristovski**
The Electrical and Computer Engineering Department,
The University of Michigan, Dearborn, USA

***Abstract - In the competitive world of software development, it is important to have an efficient, streamlined process. Without one, the risk of failure and losing money are prominent. A software lifecycle helps to keep a project on time within budget while sustaining the highest possible quality. This paper looks to refine each phase of the previously proposed lifecycle, ResPCT, including what should be delivered after each phase. Then, to test the effectiveness of the refined method, a web application that ResPCT users can use to act as a tracking system to help guide them was developed.***

***Keywords - Software Life Cycle, SDLC, Software Engineering, Design, ResPCT, PSP, Web Application Development***

## I. INTRODUCTION

ResPCT began as a personal software process (PSP) for our own use. We were having difficulty using many of the currently available lifecycles because none were made for one person teams. They also came with tools that had functionality that would never be used or the prices to use even basic features were very high. To find something that was easy to follow and help keep organized was sought. We started looking for tools that could help discovering that we weren't the only one facing this problem. In fact, many small teams don't ever bother adapting a software process, with the thought that processes are established with only large teams in mind [6].

ResPCT was originally proposed in 2013 [8] for the sole purpose of being applied by individuals and small teams. While that is still the intent of ResPCT, the original structure was lacking in anything more than a basic foundation. It set to act as a guideline whenever a new project was started [8]. Currently, ResPCT is comprised of four common stages found in nearly every lifecycle: Research, Prototype, Create and Test. The intent of this paper is to completely develop each phase, discovering deliverables and priority of phases along the way.

This paper is divided into the following sections, each of which is important in the refinement and revision of ResPCT. Section 2 will present the results of a questionnaire given to professionals to discover what is important in a software lifecycle will be discussed. Section 3 will present three popular and impressive lifecycles that will be compared to supplement the information retrieved from the questionnaire. From there, ResPCT will be revised and applied to verify if it is a success or failure. Section 4 will present the building of a web application to be used in parallel with the methods of ResPCT that will help uncover any missing pieces of the process puzzle. Finally, section 5 present conclusion remarks.

## II. QUESTIONNAIRE

To start this project, a survey of 35 statements and questions was given to 20 professionals, each varying in experience and role in the workplace. The roles of the question-takers consisted of: engineers, Quality Assurance Teams, UX/UI Designers and Project Managers. The questions were a combination of multiple choice and short answer as shown in appendix A. The participants were then chosen at random to have one-off conversations about lifecycles.

The questionnaire started with simple opinion statements relating to each phase of ResPCT, with the intent of finding what, out of each phase, was most important. These questions included some extreme statements and some phase-combination statements. Question takers were asked to give an answer based on a 5 point scale (Strongly Agree, Agree, Neutral, Disagree and Strongly Disagree). Then, these short answer questions were asked:

- Are people using processes?
  - If so, what processes are being used?
  - If not, why aren't they using a process?
- What parts of a process are important?
- What makes a process good?

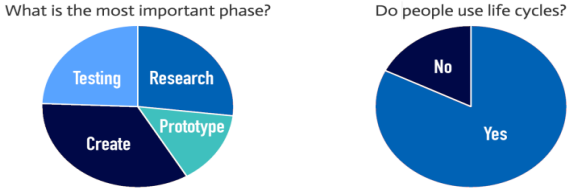



Figure 1. Questionnaire Results.

We were curious to see if professionals were using their own personal process or if they were going off the processes they were using at work. Based on the questionnaire, it appeared that many people were in fact using their work's process, however, in their own ad hoc ways as shown in figure 1. There even appeared to be mixed emotions about the way a lifecycle was being used. Some people were intent on following their chosen lifecycle to a tee while others changed or adapted it based on their circumstances.

For those developers that were not using a formal process, their ad hoc PSP still included common steps, like requirements gathering and, of course, implementation. For instance, one developer preferred to start off by looking through the requirements of a project, then moved on to simultaneously writing and testing code. Another developer preferred to gather as much information as possible, and then develop a plan to reach the ultimate end goal.

To learn more about how a process is selected within a business, we chose participants at random for follow-up interviews. In a number of cases, a process was established based on the style of their current manager. In other instances, a process was established by the company as a whole. In all follow-up interviews, the process being used, regardless of in personal or work related projects, were all tailored in some way. This is called Process tailoring. In fact, a study in [13] showed that most companies use some form of Process Tailoring.

This idea of Process Tailoring stood out because one of the ideas behind ResPCT is the ability to customize it. If large teams are customizing more formal processes, then this must be something that is important for a lifecycle to succeed.

### *2.1. Important Parts*

Based on the questionnaire results, Creating and Researching were the two most important phases in a lifecycle, followed closely by Testing as shown in figure 1. It was no surprise that many people chose more than one of the phases as important. There was, however, barely any mention of Prototyping; in fact, some participants decided to skip that section completely (their lack of response was added to the Neutral column). This result was not anticipated, considering we're in a period where usability and User Experience are thriving. A majority of participants agreed that the *Design as you go* mentality was more efficient than creating pixel perfect designs.

Expectedly, however, there were conflicting views on the importance of phases based on the person's role. One developer felt that spending excessive time on Requirements gathering was time taken away from the project itself. *"Customers don't know what they want more than developers know what they're going to build."* Conversely, a Systems Analyst argued that regardless of what process is being used, time should be spend on requirements and design. While neither opinion is wrong, it madeus aware of the fact that people will use lifecycles differently, and that adaptability is a must have for a process to work.

One thing which stood out was the overwhelming response regarding the testing phase. Each participant agreed that testing should be done throughout the entire process, not put off until the end.

Outside of phases, the ability to cycle through a lifecycle was important to many of the participants. One Software Engineering Manager stated that they preferred not to use phases only once, but rather continuously cycle through all phases during development.

### *2.2. A Good Process*

Many people were in agreement that a software lifecycle should be used as a framework for completing a project. According to the short answer responses from the questionnaire, a lifecycle should help in making a project more manageable and organized. It should be used as a tool, not as a strict way of life. The results of the questionnaire and one-off interviews reminded us that development projects aren't all the same. Each is unique and may require different aspects of a lifecycle. To have a good process means having the ability to change when needed and adapt to whatever situations may arise.

## III. REFINING ResPCT

Generally all lifecycles share the same common phases: Research and Requirement gathering, Design and Prototype, Implementation, and Testing. What makes each process different is the way these phases are ordered, and what occurs during each phase. Keeping the previous questionnaire in mind, we looked into three popular methodologies to gain even more insight as to what makes a good process. For the following sub-sections, special attention was payed to what phases each lifecycle included, along with the techniques that were involved.

### *3.1 Agile*

Agile software development is a group of methodologies centered on iterations, adaptability, and collaboration [3]. Since many agile methodologies included stages similar to ResPCT, we concentrated solely on the 12 principles that were created from the Agile Manifesto [1, 5] in 2001. More specifically, the principles that were directly related to software development and lifecycles were looked at. Of the 12 original principles, the following five were considered.

#### *3.1.1 Early and continuous delivery of working software*

The trend in software development appears to be going in the way of iterations and continuous delivery. Seeing as Agile is aimed at medium to large teams, this technique makes sense. Which leads to the question, can iterations be applied to small teams with small projects? The simple answer is yes. While the iterations may not be as complex as they would be for a larger team, they would still be helpful regardless of team size.

The basic idea behind iterations is to produce a product throughout a period of cycles or iterations. Each iteration should consist of complete and working code including the requirements that were designated for that iteration. For smaller projects, each iteration could be a daily, weekly or even bi-weekly occurrence. Similar to how a larger team uses iterations; the goal is to learn more about the product being built.

*3.1.2 Welcome changing requirements, even late in development*

Usually, changes in small projects are few and far between, as the goal of the product being built has already been established. However, when change does happen, it could be potentially dangerous. While it is believed that it is important to keep an open mind to change, one has to wonder if it would be more beneficial to focus on gathering the requirements up front and keep change from happening whenever possible.

*3.1.3 Working software is the main measure of progress*

Once again, this concept goes back to the idea of iterations. Since small projects are usually completed in a matter of weeks, having working software may be an unreliable measure of progress. In some instances, working software could be up within a day, depending on the project. Progress for small projects could be better measured by the number of tasks that have been completed. This would, at least, give some inclination as to the status of the project, and whether or not it's going to be on time.

*3.1.4 Able to maintain a constant pace*

The problem with finding a process for small projects is the lack of appropriate time management. Most small projects are completed quickly. Wasting time on a long process could hinder productivity, hence becoming costly. Being able to quickly move from one entity to another with little lag is a mark of a successful process.

*3.1.5 Simplicity*

Breaking down requirements into smaller, understandable pieces is a must when it comes to simplifying work. The ability to fully understand the problem as a whole will decrease the amount of potential rework.

### *3.2. Feature-Driven Development*

Feature-Driven Development (FDD) is a lightweight method of Agile [14]. Similar to Test-Driven and Design-Driven Development, this method dedicates itself to one aspect of a project: features. The main responsibility of anyone doing FDD is to compile a list of features for a given project based on that project's requirements. Then those features are implemented, one at a time. The goal of FDD is to break the project's features into small pieces that can be delivered iteratively over time. Each iteration, like any agile methodology, should be fully functional.

FDD is appealing because of its focus on breaking down features into bite sized pieces. This benefit, along with the use of iterations, would make getting working code out faster more feasible.

### *3.3. Waterfall*

Waterfall is the most traditional of the three software lifecycles and has been the foundation for many other lifecycles. It consists of five or six phases, depending on the version being used. For the sake of this paper, we will look at the five phase model. The phases in question are: Requirements, Design, Implementation, Testing and Deployment and Maintenance. In the Waterfall lifecycle, phases are done sequentially, with little to no iterations. In the more traditional method, there are no iterations and no return to previous phases [2].

It is important to look at the Waterfall lifecycle because of its extensive history. While it may seem archaic now, it was still used prominently for decades. In fact, still currently being used by many companies. Its focus on completing each stage first minimizes the potential for error, but also little room for change. This may be beneficial for small projects where any changes could be a potential catastrophe.

### *3.4. Comparison*

While each process above has their individual benefits, they still all share common functionalities. The ability to iterate, for example, is most important for Agile and FDD to succeed. Even in current versions of the Waterfall lifecycle, users have the ability to do, although fewer, iterations during a project. Working smarter, not harder, is another shared trait. Agile and FDD focus on breaking the project down into manageable pieces, while the Waterfall lifecycle looks to gather as much information as possible to reduce rework. Finally, they all put substantial effort into requirements and implementation, making them both a high priority. These commonalities, along with the research from the previous section, will be used in refining and revise ResPCT [8].

### *3.5. ResPCT*

A lifecycle should be considered as a tool, not a way of life. To make a great tool, it takes practice to become streamlined. Each phase within ResPCT should be seen as practice, or a number of tasks that need to be completed [5]. ResPCT as a whole can be defined as a Method, or a system of combined practices [8]. Keeping this mindset will help the user move from one phase to another with little interruption.

ResPCT will continue to consist of the four original phases. Research will be used to gather all required information for the project. Prototyping will deliver a design built from those requirements. Creating will give life to those requirements, and Testing will make sure those requirements are met and delivered. ResPCT can be furthered summed up as follows: Requirements should be broken down into Tasks, and then each Task should be linked to one or more Test as shown in figure 2.

Changes were considered after looking at the results of the questionnaire and the comparison of the three lifecycles above. The first change to ResPCT was adding a Release phase. While ResPCT covered all the phases from conception to testing, there was no criterion on when a product was ready to be released. Another important change to ResPCT was adding the ability to iterate. Finally, different workflow tracks were added to build on the customizability of ResPCT. The following sub-sections will go over these improvements, as well as the finalized deliverables for each phase of ResPCT, and how ResPCT can be applied to small teams.

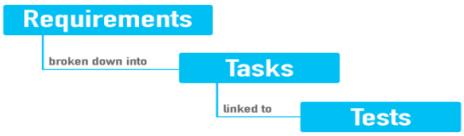



Figure 2. ResPCT Principle.

*3.5.1. Research*

The goal of the Research section is to gather as much information about the project as possible. During this phase, you will have three deliverables:

- Problem Statement
- Requirements
- Scratch Pad

At the beginning of a project, it is important to state the problem that is to be solved. This will keep the end goal of the project in mind. It is intended to be kept general, but specific enough to not be vague. For example, let's take a system that stores information about groceries at home. *Home inventory* may be too vague, while *I need something that will tell me all the food items I have at home in my pantry* may be too specific. Ask yourself, "What is the problem?" In the case of the home inventory application, a good Problem Statement may be *buying duplicate items at the grocery store*, or *not knowing what groceries I have at home.*

There should be a great deal of effort put into gathering requirements. While you can iterate through ResPCT, the aim is to avoid changing requirements. Obtaining as much information as possible upfront will avoid rework. A requirement should be a description of something that the user or product owner wants, and must be testable [16]. Continuing on with the Home inventory example, a good Requirement may be "The Ability to scan a product with a UPC Scanner Gun".

The final deliverable, Scratch Pad, is used as needed. This section can be used to jot down ideas, including design, interactions, or coding examples. It's a place to keep those "bright ideas", with the goal being to keep a project organized.

*3.5.2. Prototype*

Prototyping should consist of all things design: system, database, user interface and interactions. Prototypes can be many things, including hand-drawn sketches or pixel perfect Photoshop documents depending on which track you choose. For a more traditional process, designs should be completed before moving onto the implementation phase. For a more modern approach, designs could be working code rendered in a browser.

*3.5.3. Create*

In the Create phase of ResPCT, implementation is the ultimate goal. There are several things that should be considered during this phase. First of which is to always use source control [10]. Source control is a good way to keep versions of code in the instance of having to revert back to working code. Also, testing and coding should be done simultaneously. Testing in any process that is held off until the end is an inherently flawed process.

The deliverables for the Create phase consist of URLs for the development environment being used, the URL for the source code repository and a list of Tasks for the project. Like in FDD, Tasks should be broken down into small pieces that are clear and concise. Taking the Requirement for the Home Inventory project, we can break it down into the following tasks:

- Create a database to store Item UPCs
- Write a program to retrieve UPC scanned input
- Write a program to store UPC scanned input into database

*3.5.4. Test*

As stated in the previous section, it is beneficial for testing to be done in tandem with implementation. Tests should also be run early and often [10]. In the event that a test fails, the code should be changed to pass the test, not the other way around. Tests should be written to accompany Tasks. Each Task from the previous phase should have at least one corresponding test connected to it. Having more tests will provide better coverage for the system, making it less likely that users will find bugs in production.

In the Testing phase ResPCT will have two deliverables: Tests and Bugs. Every test added should include a name, description, testing steps, the expected and actual results and the status of the test (whether it's pass or fail). Example Tests for the Task, "Write program to store UPC scanned input into database", may include:

- Testing the database connection,
- Checking to see that input was retrieved from the UPC scanner, and
- Verifying that information was stored in the database.

Tests should be written when Tasks are written. Once a task is implemented, the tests accompanying that task should also be implemented and ran. If multiple tasks are completed in one sitting, the tests should still be implemented; however they can be run either sequentially or separately.

In the event that a bug is found, it should be documented. Each bug should include a name, link to the page it was found on, a description and a status (whether it is an open or closed bug). Once a bug is documented, there should be a task added to fix that bug (if there isn't already) then, it will need to be retested and marked as open or closed.

*3.5.5. Release*

When using the more traditional ResPCT workflow, a product is ready for release when all Requirements have been meet, all Tasks have been implemented, all Tests have been run and passed and any bugs found have been fixed. It is important to verify that requirements have been met, as that is a mark of a successful product. When using iterations, smaller releases will occur, but once all requirements and implementations have been finished, a larger release will follow the above guidelines.

*3.5.6. Tracks*

During development of the web application, while trying to follow ResPCT sequentially, we found ourselves continuously going back and forth between Prototype, Create and Test. This helped us realize that a two track system would benefit ResPCT. Khan [1] introduces the idea of heavyweight (HW) and lightweight (LW) methodologies. HW methodologies are more traditional and include sequential steps, while LW methodologies are focused on being adaptive and simultaneous. With that in mind, ResPCT was split into two tracks: Turtle and Rabbit.

*3.5.6.1 Turtle*

The Turtle Track can simply be described as slow and steady. It is geared toward the more traditional developers, those which are more information drive. In the Turtle Track, ResPCT is completed sequentially, moving one phase at a time; however, Create and Testing should still be completed together. With the Turtle Track, there should be fewer iterations, with more focus on getting as much information, like requirements, as possible.

*3.5.6.2 Rabbit*

The Rabbit Track is faster paced, with working code being the main focus. In contrast to the Turtle track, Requirements are gathered first and foremost, and then design, implementation and testing can begin. The Rabbit Track is for developers that prefer to use coding as a form of design. Here, there should be more iterations completed, with a design as you go mentality.

*3.5.6.3 Hybrid*

Customizability and Adaptability are typically important factors when choosing a lifecycle. With that in mind, ResPCT was generalized enough to be moved around and reused in any way that seemed appropriate. For example, if the person is more Test-Driven, they can put more of their effort into making sure the project is thoroughly tested. If they are Design-Driven, they can focus on completing a well thought out design before starting any other phase. While ResPCT still has its own deliverables, more can be added or removed based on personal preference.

*3.5.7. ResPCT and Small Teams*

Take, for example, a team consisting of three people: a project manager, a designer and a developer. Each person has their own roles and a phase (or two) of ResPCT to complete. A project manager's sole responsibility will be to gather the requirements for the project and pass them on to the designer and developer. The designer would be responsible for creating the design of the application, while working with the developer to create Tasks. Once all Tasks have been created, the designer and developer will implement these tasks. The developer will have an extra step to complete by creating and running the tests. All three will be responsible for reporting bugs found in the system.

Another example of how ResPCT can be applied to small teams would be to look at a team with two sub teams: a backend team and a frontend team. Each team has their own responsibilities, along with each sub team having their own requirements, design, tasks and tests. With that in mind, one project would be split into two separate ResPCT workflows - one for backend and one for frontend. Each team will be responsible for iterating through the phases while keeping the big picture in mind.

*3.5.8. Iterations*

An appealing aspect of Agile and FDD was the availability of iterations. Considering this, the capacity to iterate was added to the ResPCT lifecycle. To effectively use iterations, the Tasks and Tests sections will need to be modified. For every iteration, the Tasks and Tests should only be those needed for that specific iteration. Once an iteration is released, new Tasks and Tests should be added. An iteration is ready for release once all tasks for that iteration have been completed, all tests have been run and passed and any bugs found during that iteration have been fixed.

## IV. APPLYING ResPCT

Once ResPCT was fully redefined, it was tested to verify its success or fail rate. To do this, a web application was developed. This web application was to be created for ResPCT users to organize and track their projects, similar to how agile users have Jira. The main challenge of creating a tool for users is making it efficient and lightweight, while still following the ResPCT workflow. The following sections go over each phase of ResPCT, and any difficulty following the new process was documented. Any problems were then prioritized, and added into the process.

*Research*

We started with this project following the ResPCT workflow, focusing on finding the problem statement and gathering as many requirements as possible. The problem that we were trying to solve for this application was *organizing ResPCT projects*. Requirements for this application included:

- having a database dedicated to each phase of ResPCT
- the ability to add or remove projects
- the ability to add or remove deliverables within those projects

Once we had gathered the remainder of the requirements for this project, we jotted down a few ideas that we had for the design and interactions. We wanted the interactions to be catchy, so we made a note to look into jQuery and modal interactions. We also knew that we would be using a MySQL database, which meant an added note about looking into database design.

### *4.1 Prototype*

Once we got to this phase, we started having trouble following the ResPCT workflow as it was (at this point, ResPCT didn't have the option of customizable tracks). We realized that with the time constraints of the project, we wouldn't have enough time to do a pixel perfect mock-up, so instead we started to create the pages within a browser before we had decided on a final design. We also had simultaneously created the databases for the project, choosing to forgo a more formal database design.

At that point, we went back to the refined process, and reevaluated the workflow. After some research, the tracks for ResPCT were established and the Rabbit track was used moving forward. Figures 3, 4 and 5 show the final interfaces of the application.

### *4.2 Create*

As previously mentioned, we were designing, implementing and testing simultaneously at this point. While our requirements were complete, our task and test lists were not. Whenever we thought of a new task, we would write it down, add additional tests, and then begin implementation. This is where we struggled most during the project because we didn't have a tool to help keep us organized and on track. After the application was built, we went back and tried the process again with a smaller project. Having the tool made a significant difference with helping keep track of tasks and tests.

Originally in the create phase, Tasks were listed as Features, however we felt that Features made it appear more about functionality, which is what the Requirements were for. Moving forward, they were changed to Tasks, and focused on what needed to be done to complete the project.

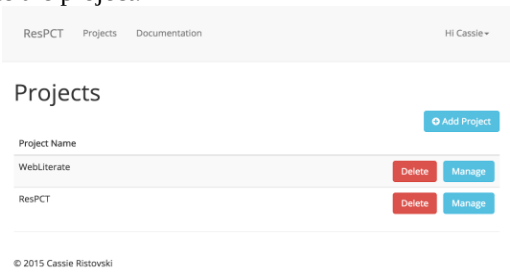



Figure 3. ResPCT Home Page.

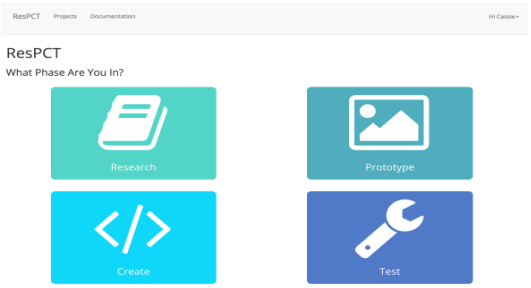
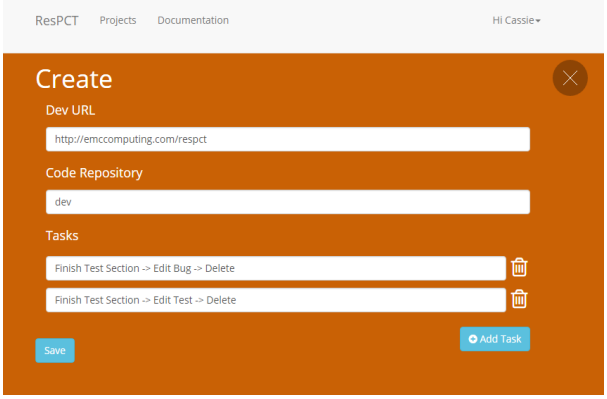



Figure 4. ResPCT Project Page.



### *4.3 Test*

At the point of conception, the Testing phase was purely about the tests. Any bugs would be added to the task list, with no additional information. This didn't work out so well, since we were losing track of what was a bug and what was a task. Hence, to help differentiate, bugs were moved to become a part of the Testing Phase and included some much needed bug information. Once the application was finished, we realized that there wasn't anything telling us if we were ready to release. Once this option was added, ResPCT was updated with the new findings and completed.

## V. CONCLUSION

The main goal of a lifecycle is to help keep a project on track. The tools that accompany a lifecycle should help make a project more efficient and easy to follow. Above all, the lifecycle and tools should be accessible and usable for teams of all sizes. ResPCT was originally built to help individuals and small teams, and has been redefined to become a more efficient and sustainable lifecycle. It will always be free and open-sourced, as it is intended to be reworked, rearranged and reused to fit any project or personal style.

## REFERENCES

[1] M. A. Khan, A. Parveen, and M. Sadiq, "A Method for the Selection of Software Development Life Cycle Models using Analytic Hierarchy Process", *IEEE International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*, 2014.

[2] N. B. Ruparelia, "Software Development Lifecycle Models", Hewlett-Packard Enterprise Services, *ACM SIGSOFT Software Engineering Notes* Vol. 35, No. 3, 2010.

[3] P. Abrahamsson, J. Warsta, M. T. Siponen, and J. Ronkainen, "New Directions on Agile Methods: A Comparative Analysis", *IEEE International Conference on Software Engineering*, 2003.

[4] S. L. Spraragen, "The Challenges in Creating Tools for Improving the Software Development Lifecycle", *ACM SIGSOFT Software Engineering Notes* Vol. 30, No. 4, 2005.

[5] F. J. L. Hinojo, "Agile, CMMI, RUP, ISO/IEC 12207… Is There a Method in this Madness?" *ACM SIGSOFT Software Engineering Notes* Vol. 39, No. 2, March 2014.

[6] R. V. O'Connor, C. Y. Laporte, "Towards the Provision of Assistance for Very Small Entities in Deploying Software Lifecycle Standards", *Product Focused Software Development and Process Improvement (PROFES)*, 2010.

[7] J. M. Willenbring, M. A. Heroux, R. T. Heaphy, "The Trilinos Software Lifecycle Model", *IEEE Third International Workshop on Software Engineering for High Performance Computing Applications*, 2007.

[8] Adnan Shaout and C. L. Dusute, "ResPCT – A New Software Engineering Method", *International Journal of Application or Innovation in Engineering & Management (IJAIEM)* Vol. 2 Issue 12, 2013.

[9] IEEE, "1074-2006 - IEEE Standard for Developing a Software Project Life Cycle Process", *IEEE Standards*, 2006.

[10] A. Hunt, D. Thomas, "The Pragmatic Programmer", *Addison Wesley*, 2000.

[11] K. Kumar, R. J. Welke, "Methodology Engineering: a proposal for situation-specific methodology construction", Challenges and Strategies for Research in Systems Development, W. W. Cotterman and J. A. Senn, Eds, New York: *John Wiley & Sons*, 1992, pp. 257-269.

[12] T. Winograd, "From Programming Environments to Environments for Designing", *Communications of the ACM*, Vol. 38, Issue 6, 1995.

[13] G. Coleman, R. O'Connor, "Investigating Software Process in Practice: A Grounded Theory Perspective", *Journal of Systems and Software*, Vol 81, No. 5, pp 772-784, 2008.

[14] A. F. Chowdhury, M. N. Huda, "Comparison between Adaptive Software Development and Feature Driven Development", *I EEE International Conference on Computer Science and Network Technology*, 2011.

[15] C. Bast, "Hewlett-Packard's Approach to Creating a Life Cycle", *IEEE*, 1994.

[16] G. Miller, "Want a Better Software Development Process? Complement it", *IEEE Perspectives*, 2003.

[17] S. R. Palmer, "A Practical Guide to Feature-Driven Development", *Prentice Hall*, 2002.

[18] S. Ambler, "Feature Driven Development (FDD) and agile modeling". [Online]. Available: http://www.agilemodeling.com/essays/fdd.htm, 2015.

**AppendixA - Questionnaire**

| Data | Strongly Agree | Agree | Neutral | Disagree | Strongly Disagree |
|---|---|---|---|---|---|
| Before I start a project, I do extensive research. | 7 | 7 | 2 | 4 | 0 |
| It's a top priority for me to know as much as I can about a project before I begin. | 9 | 6 | 0 | 5 | 0 |
| When/if I do research, I make sure to cover all of my bases. | 5 | 12 | 2 | 1 | |

| | | | | | |
|---|---|---|---|---|---|
| I do as little research as possible | 0 | 2 | 0 | 11 | 7 |
| Research is the most Important phase | 3 | 8 | 6 | 3 | 0 |
| During research, I make sure to gather all of my requirements before moving on to the next phase. | 6 | 3 | 6 | 5 | 0 |
| I make sure that requirements are approved or accepted before starting any further work. | 6 | 7 | 3 | 4 | 0 |
| I create a mock-up or prototype before I start coding/ creating | 6 | 8 | 4 | 2 | 0 |
| Knowing the design of the product first is most important to me. | 3 | 8 | 6 | 3 | 0 |
| Designing is the most Important phase | 3 | 3 | 10 | 4 | 0 |
| I use requirements and research to back my design. | 6 | 11 | 2 | 1 | 0 |
| My design is usually just a rough sketch. | 2 | 4 | 11 | 3 | 0 |
| My design is usually extensive. | 0 | 6 | 8 | 6 | 0 |
| I usually create a fully functioning prototype before moving on to the next phase. | 1 | 2 | 8 | 7 | 2 |
| Creating is the most important phase. | 7 | 7 | 4 | 2 | 0 |
| I usually start this phase first before doing any design or research. | 1 | 1 | 0 | 15 | 3 |
| I make sure I have everything I need before starting. | 3 | 10 | 3 | 4 | 0 |
| I often refer to my research and design when creating the product. | 5 | 9 | 5 | 1 | 0 |
| My design helps me discover what's possible and what's not when creating the product. | 4 | 8 | 3 | 5 | 0 |
| I make sure my code is complete before moving on to the next phase. | 3 | 10 | 3 | 3 | 1 |
| Testing is the most important phase | 3 | 7 | 5 | 5 | 0 |
| I write tests as I go. | 2 | 6 | 8 | 3 | 1 |
| I write all of my tests after I finish coding. | 1 | 1 | 9 | 7 | 2 |
| Testing is something I rarely do. | 0 | 2 | 1 | 8 | 9 |
| I write tests first and use the tests to help create code. | 3 | 5 | 9 | 2 | 1 |
| I deliver my product before completing testing. | 0 | 1 | 2 | 6 | 11 |
| I always use a life cycle when working on a project. | 8 | 6 | 3 | 3 | 0 |
| I am strict about following the rules of the life cycle I use. | 1 | 6 | 9 | 4 | 0 |
| I don't see the point in using a life cycle; I usually just wing it. | 0 | 3 | 4 | 7 | 6 |

| **Do you currently use a software life cycle (i.e. Agile) in your personal or work projects? If yes, what life cycle are you using, and what do you think works for that life cycle? If no, what steps do you complete, from start to finish, during a project?** | **What is most important about using a life cycle during a project?** | **What benefits do you look for when choosing a life cycle to use? What are some disadvantages?** | **What is your current job title? What is your primary role at work?** |
|---|---|---|---|
| No. I like to read the spec and plan out my design first, although I do not go into a large amount of depth. Usually I write and test my code simultaneously, with a bit of extra testing whenever I think I am done. I do not make a distinction between prototyping and coding phases, since prototype code tends to turn into the final submitted version. | | | |
| Agile lifecycle; Short cycle of coding/testing, quick feedback loop. | Consistency, with a deliverable product at all times | | Developer, programming and design |

| | | | |
|---|---|---|---|
| We use a form of Agile but it's incredibly lean. We have 1 week sprints and constantly push so things are continuously changing. I think software life cycles can work, but it depends on the team and what works best for them. For us, we design as we go and change things on the fly if they're not working. I do a quick sketch and code it out because it's faster to code than to use photo shop for a pixel perfect mockup. Mockups that are more than wireframes tend to lead on people and they get focused on colors or shadows or whatever instead of the workflow. (Long story) | Don't let the process dictate the product. It's a tool not a way of life. If it helps, great, if not, don't use it. | Whatever works for you? I think the only wrong answer is to say there is one answer. Agile works when everyone has an agile mindset. Some teams love a strict process and follow it to the letter. Others change and adapt according to the circumstances. Some are a bit of both. | UI/UX Designer - All things wireframe-y, CSS SASS or LESS, and front end architecture. |
| The current project I am on uses Agile. The project lifecycle has included a 1. Research phase - in this they included estimates for development 2. UI mockup approval requirements from product owner/ client 3. Development/UI implementation 4. UX testing 5. Recommendation/rework of design 6. Demo of product 7. Any additional changes 8. Launch. Along the way there is discussion about timeline- whether demo date needed to change. | The research phase is most important, especially estimates for development. Estimates are important since they could determine the timing of test, demo, and release. In certain cases outside research when the team is unfamiliar with the product. The outside research could also help with technology needed to fulfill product owner requirements if needed. | It is not just about getting a product out the door quickly, but the quality and usability it retains. If the multiple phases lead to a successful product then they are necessary. There are times when the product owner veers from the goal of each phase, sometimes back-peddling the team. That is when it is important to remind them of timeline; demo date and release date. | UI stylist. To implement mockup specifications provided by the designer and product owner using CSS/LESS. |
| We use Scrum but honestly most of the time extensive requirements gathering before a project is a waste of time. Customers don't know what they want any more than developers know what they're going to build. | | | .Net Developer |
| No | Ensuring that all checks and balances are in place. | A logical and efficient workflow that covers all necessary tests is best. This can, however, slow down the production process. | Production of content |
| Officially I use Agile at work. The steps of the lifecycle are always basically the same with the only difference between the models being time spent per step, if any should be repeated, and how to handle eventual defects. | | It doesn't matter, they're all the same. The most important thing to do, regardless of model, is make sure you spend more time on requirement gathering and design. Also, testing should occur and be considered through all phases, not just put off until the 'test phase'. | System analyst. My primary role at work is to be given a request for an addition or modification to an existing system or system component then work through all phases of the development cycle for said change/addition. |
| I guess by "life cycle" you mean "development methodology"? I use agile in work projects, and try to use it in personal projects too when possible. I prefer not to use phases, like you list above - I prefer to continuously cycle through all phases many times during development. | It gives you a framework to make the project more manageable. | Easy to follow, allows rapid response to change, gets me to working software as quickly as possible, more code, less other things. | Software Engineering Manager |

| | | | |
|---|---|---|---|
| Analysis, Planning, Design, Build-Test, Test, Test, Deliver, Support | Following a process ensures we deliver quality products | Flexibility and alignment with company culture | Senior System Analyst responsible for fleshing out requirements, documentation, testing, etc. |
| My company uses the Agile Scrum model. I think that this method keeps the team able to respond to changes in the market or in the product well, and produce fully functional components very quickly. By doing work in sprints, we are able to focus on one small part of a larger problem at a time and do a thorough job in the design, planning, and testing of that component. This keeps the team from getting lost or distracted by other details, and allows us to be very productive. We used to follow something more like a waterfall method, which had pretty bad results for us. By the time we would get around to coding something that was a later part of the design plan, it was often no longer relevant and needed to be changed. The testing and fixing phase was also a mess, because our testing results often required changes to design, which were very costly to fix and often delayed releasing the product to market. Our current agile method pretty much prevents this from happening. The only weakness is that we have had to be careful making sure there is some consideration given to the big picture for the software architecture in the beginning. Since the Scrum methodology does not support having a design phase for the whole product, this could create issues with components working together when building a complex system. So we've modified the method a bit so that we can address this weakness. | It is important for to have a standard process so that the team understands what is expected of them at any given time, and allows for better focus and communication. It also allows us to be more efficient overall by solving problems and catching errors with the lowest cost possible when we are able to do so in early phases. | My company provides a web and mobile based SAAS product, so it is very important for us to be able to quickly develop a fully functional product so as to provide continuous updates. When we produced a Windows based application with a one-time purchase, this was not as much of a concern for us. A long software lifecycle would support releasing extensive functionality in yearly updates to the software, which were re-purchased, and the costs of development were covered fine with the model. However, a long lifecycle now proves to be too costly for our business. With a subscription-based service, we need to constantly make sure to fix any issues as quickly as possible to prevent losing users, and we need to add functionality to remain competitive. Being constantly concerned with maintaining users makes the costs of development much more clear, and having long research, design, etc. phases creates more time where we are not benefiting from having important customer-pleasing functionality out and available to our users. I believe that since many software companies are moving to a SAAS model, this would be very important to a majority of software development companies going forward. | Product Manager. I am in charge of all research to determine what needs to be added/changed in our products, and defining the requirements for these items. I created basic functional, UI, and performance requirements. I produce mockups of new components for multiple interfaces, and then I pass the requirements along to our software architect who does the database design and software architecture design. In our SCRUM model, I also act as the product owner for 3 coding teams for coordinating sprints. |
| General concept and goal, gather as much information as I can, find a path to reach goal, plan, time-line, resources required, time required, pre design, detail design, design test, making prototype, design validation, pre production, production test, validation and final production. | | | Design Engineer. To design new products following product development path or cycle |