**Back Yard Adventure Agile vs. SDLC**

Morrese D. Morrison

Pierce College

Computer Science 302: System Analysis & Design

Study.Com

October 7th, 2023

## Back Yard Adventures Agile vs SDLC Introduction

Software development methodology is a crucial part of the development of a software or system product. Successfully identifying and selecting the appropriate methodology for a project will have a major impact on how successful that project will be. Picking the incorrect model can cost you precious resources such as time, money, and personnel. As such, it is imperative to thoroughly understand each methodology in regard to their inner workings and their intentions. With that knowledge and understanding, one will be able to make the correct choice when embarking on their development journey.

**Agile vs SDLC**

The two main methodologies that are frequently utilized is the agile methodology and the software development life cycle methodology. These two main methodologies are used frequently in a business environment. Both have their intended best use cases as well as their pros and cons. But on a macro level, both of these methodologies although similar have fundamental differences in their overall approach. Due to this, it is important to thoroughly review these differences before choosing which model you would like to utilize.

But before the differences are discussed, it is imperative to understand the foundational philosophy between both methodologies. “Agile is an iterative incremental software development approach. In Agile, the product is delivered in fixed-term time intervals by the cross-functional self-managed team” (*Bugajenko, n.d*). With agile development, this methodology follows six stages to produce a part of a product. These six stages include analysis and planning, requirements definition, design, development, testing and operation and maintenance. Once these stages are completed, that part of the project is released to the customer or various stake holders. Then after that, those same six stages are repeated again for the next part of the product to be completed then released in addition to the previous part of the product. This continues until the final product is fully built. You can think of it as you being a general contractor working on building a staircase. After you build a step, you show the homeowner. And you continue to do that until that until the staircase is completed. This in essence is the philosophy of the agile methodology.

This differs a bit from the software development life cycle methodology. The software development life cycle methodology takes the same six stages we mentioned for the agile method but applies it to the entire project as a whole. “The stages do not overlap, so the previous stage must be completed and signed off by the stakeholders before the next stage can begin” (*Bugajenko, n.d*). So, these six stages are applied for the entirety of the project, instead of just parts. Back to the general contractor analogy of you building a staircase. Instead of the agile approach where you showed the homeowner each step after it was built, you now only show the homeowner the completed staircase once you are completely finished.

As similar as both of these methodologies are, what are the main differences that need to be accounted for when choosing between them? There are four main differences that will be emphasized that will draw the distinction between both methodologies. These distinctions include philosophy, objective, customer involvement, change and quality control. From a philosophical approach, the software development life cycle “follows a predictive approach, with detailed planning and defined user requirements” while “ agile follows an adaptative approach based on quick feedback exchange and continuous design improvements” (*Bugajenko, n.d*). The software development life cycle in a sense is more rigid as everything is detailed and planned beforehand while the agile approach is based on quick development. The software development life cycle in this case would be best applied of the requirements of a project are static and well defined. With agile, this would best be applied if that same project had a dynamic requirement as opposed to static. The agile approach would be better equipped to handle that.

The second main difference deals with the overall objective. The software development life cycle’s main objective “risk minimization achieved by thorough requirements analysis, while in agile it is quick product delivery” (*Bugajenko, n.d*). This is an important difference. Based on the project, if risk needs to be minimized as much of possible, the software development life cycle should be applied. The development of a critical infrastructure system is an excellent example of this. This system needs to be thoroughly assessed and tested to mitigating as many faults as possible. If not, these faults could cause infrastructure vulnerabilities which can result in the population not being able to access water, electricity, or gas. Agile in this case would not be appropriate as the focus on quick delivery without thorough testing and assessing will almost certainly introduce a product that will multiple faults that will cause issues to the environment and the population.

The third main difference is customer involvement. With the software development life cycle “the customer is involved only in the beginning of the project when the requirements are analyzed and documented. In agile, the customer representative continuously participates in the development process” (*Bugajenko, n.d*). This is also an important difference, depending on the product, constant involvement may or may not be needed. In the case of a project where the stakeholders only care of the product produces the desired requirements, and they do not care how it is done, then the software development process will be appropriate. If this is not the case and customer or stakeholders require constant updates and to give feedback, then agile will be appropriate. Of course, these levels of involvements can be interchangeable and used in unison if applied corrected.

The fourth difference is change is also another major difference between the two. Change in regard to the software development life cycle is rigid. Once the requirements are set and agreed upon. It is best that they are not changed. If they do need to change this can cause cause loss in monetary and time resources due to the methodology in general being static. Agile in other hand is excellent with change due to its methodology being built upon flexibility with the end product in mind. Again, each use case will depend on these methodologies being applied appropriately.

The final main difference is in regard to quality control. Quality control with the software development life cycle “is difficult, because the testing is performed only once the whole development stage is finished” (*Bugajenko, n.d*). However, the agile method preforms “testing after each iteration, ensuring permanent control and lowering the cost of fixing a mistake” (*Bugajenko, n.d*). This of course has pros and cons in itself. Depending on the size and intricacies of the project, either approach may be appropriate.

With the differences discussed, what are the overall pros and cons of the software development life cycle and the agile methodology. There are three main pros and cons to the agile methodology. We will address the pros first. The pros to the agile methodology is that this it allows for “faster implementation of changes”, “more flexibility for changes and better communication with users and greater user satisfaction” (Deborah, n.d). Because of the quick developmental sprint of agile, feedback and change request can be built and released to stakeholders or customers. This ultimately creates overall flexibility for change management. This also ties in with the constant customer communication between the development team and the stake holders. All of this culminates with the overall speedy production of said project. example of this would be the constant creation and iteration of a new website. The website can be created initially and receive feedback and certain change requests from the user base. The development team would then work on those requests that are deemed feasible, the process will repeat until the website and all of its functionality is completed.

The cons with the agile method is that it is “Difficult to assess the amount of effort and resources required”, there is “less emphasis on creating documentation and users are required to test on an almost continuous basis” (Deborah, n.d). Back to the website creation analogy, with each change request that is received, there isn’t a set time that is associated with that request in regard to development. It could range to from a few days to a few weeks. One request may be to change the font of a page, another request could be to utilize various API calls within the website to receive and display data. Also, due to the fast past developmental philosophy of the agile methodology, the emphasis is always on creating and shipping the product as quick as possible. With this mindset, proper documentation is not properly created and organized. This can cause issues with new team members that are joining the development team or the simple fact that a point of reference on how a module is created and configured is not available. This can cause issues with troubleshooting as time will be wasted on relearning that module and its various dependencies. This of course also leads to constant testing if requests are constantly being pushed out. This can easily take up resources as well as each module or part needs to be tested before being shipped.

Now that we have discussed the pros and cons in regard to the agile methodology, how about the software development life cycle methodology? There are two main pros and cons that are applicable to the software development life cycle. One pro is the “clarity of requirements”( Shikht, 2022 July 6th). Due to the project being heavily planned in the requirements phase, every aspect of the project is covered in detailed and recorded for reference. This clarity of requirements creates a measurable milestone for the development team aim for. The second pro is that a project has a “reduced risk of going off the track” ( Shikht, 2022 July 6th). This of course is extremely possible and happens quite often. If a project does go off track, the overall resources needed for the project will increase and ultimately, the end product may not satisfy the initial

requirements agreed upon.

In regard to cons, the first main con is that “if different unexpected circumstances complicate the development process, that can lead to more complications in the future” (Shikht, 2022 July 6th). For example, if a certain technological platform that was initially agreed upon, ultimately does not fulfill the development team’s expectation, then that platform would have to be replaced with another one. This ultimately will grind the development to a halt as a replacement platform would need to be identified. The second main con is that “performing testing at the end of the development process can slow down development teams” (Shikht, 2022 July 6th). This can be an issue with extremely complex projects, the testing phase may in fact take up the most time of all of the staged depending on the number of issues found. Again, this also slows down the overall product delivery as any bugs that are found will need to be addresses before that product can be released” ( Shikht, 2022 July 6th).

**Software Development Life Cycle Models**

Within the software development life cycle, there are “models”. These models are versions of the software development life cycle. But these versions all adhere to the fundamental philosophy of the software development life cycle in their own own way. You can alike it to a tree. The tree is the software development life cycle, and the various branches on that tree are the different software development life cycle models. These different models include the water fall model, the v model, iterative and incremental model, software prototyping model, spiral model, and the rapid application development model. Just as before, each model has their own pros and cons.

The waterfall method has pros such as being “easy to manage due to the rigidity of the model” and having “well understood milestones” (*SDLC - Waterfall Model*, n.d). This can easily improve the overall effectiveness of the development of a product utilizing this model. With a clear goal clearly defined as well as the overall tasks associate with the project being static, this eliminates any possible confusion by introducing new tasks that may not have been vetted. Disadvantages include not being not being able to “accommodate changing requirements” as well as the fact that “no working software is produced until late during the life cycle” (*SDLC - Waterfall Model*, n.d). With the rigidity of this method, introducing new requirements or changing existing ones might effectively restart the entire project. Also, the end product is not available until the later stages of the model. An example of this would be a console video game. The game has to be somewhat fully completed to be able to be available for purchase for consumers. But while it is being developed, it is only available to the development team working on that video game.

The v model is often referred to as the verification and validation model. That is because each phase has an accompanying testing phase. The various pros involved with this model include “easy to use” as well as being “high quality” due to the rigorous testing of this model (Wamicha, n.d). So that means within the five phases that are included with the v model such as business requirements specification, system design, architectural design, module design and coding, there is an accompanying validation phase. Fairly easy and step by step concept. However, there are direct cons to both of these pros on the other end. The counter argument to these pros is that again this model is not “flexible” and it can also be “expensive and time consuming” due to each phase requiring to be validated (Wamicha, n.d). An example of this would be the development of a new feature to a program. That feature would have to move through all five phases in addition to their testing phase. Which essentially doubles us the total number of phases, which can be resource intensive.

With the iterative model a “developer never begins with a full set of specifications but takes smaller pieces of code and builds upon each completed code to form the final version of the software” (Ransom, n.d). The phases involved with this model include a design phase, build phase, testing phase and implementation phase. Every time these phases are completed, a module is released by the developer, building upon the previous module released. The pro’s involved with this is when a “customer needs the product within a short time frame” and for the simple fact that a working version of the software can be made visible to stakeholders (Ransom, n.d). One good example of this is a development of an enterprise program system that will need to handle payroll, production, and invoices. A developer may first build the frame of the system, then add each of these modules on top of each other. Disadvantages include high usage of resources as well as the challenge of completing a certain number of tasks within a short burst of time. Back to the example, these modules may be intricate as they may have dependencies on each other. If only a certain amount of time is allocated on the development of the module, certain tasks within that module may have to be pushed back to a later iteration. This can add to the overall tasks that needs to be completed and can affect the usability of the previous completed module.

The prototyping model is model that can be effective if utilized properly. The prototype model deals with constructing a working prototype of the desired product. This prototype can be reworked and redesign as much as possible until a version that is produce fully satisfies all requirements. One of the pros that this model has is that it heavily involves the customer. The advantage with this is that you have live and constant feedback. This helps with the refinement of the product in real time. The disadvantage is that sometimes, users may believe what you show them is the final version. This process has to be thoroughly explained to the stakeholders, so they do not get confused. The second disadvantage is that there is an up-front cost. Essentially you are building the final product, however that prototype may need be extensively modified or scrapped all together. If resources are not a problem, then this disadvantage will not be an issue.

If you have a highly complex software project needs to be completed, this next model is more than capable in delivering results. The spiral model is essentially a spiral with four phases. The loop starts at the identify objectives phase, moves onto the perform risk analysis phase, then the develop and test phase then finally the review and evaluate phase. “The spiral model enables gradual releases and refinement of a product through each phase of the spiral as well as the ability to build prototypes at each phase” *(What is spiral model and how is it used?* *2019, August 15).* This model also has multiple pros and cons as well. One of the first pros with this model is that it works extremely well with risk handling. This is due to the fact that with every phase “risk analysis and handling” is intertwined *(What is spiral model and how is it used?* *2019, August 1).*The second pro which is most often mentioned with software development life cycles is flexibility. However, in this case, flexibility is extremely imperative due to the high risk of the project. Due to the high probably that requirements will change after development.

An excellent example of this would be the development of space vehicles and rockets. Reason this would be an example is due to the sheet amount of risk associated with the development of such a product. People and monetary resources are fully put against a project of this magnitude to come up with a working model. This working model of course would be a prototype. This prototype would then need to be used in various atmospheric simulations and scenarios to see if the space craft will fulfill its intended purpose. On top of this, during the development stage complex requirements may change to accommodate new needs. The spiral model would be absolutely ideal for a project of this magnitude and high risk associated with it. As unique as this model is, it is not without its disadvantages. One major disadvantage is the complexity to use this type of model. Any protocols associated with each phase will need to be followed, as well as accurate documentation being created. If not, this can add to the overall time to completion for the project. In addition, the model is expensive to run due to the complex product it is trying to produce, and the other intricacies involved. Due to prototyping involved as well as the high risk associated with the model, the overall financial implications can be disastrous if not implemented correctly.

**Back Yard Adventures Recommended Methodology**

After discussing the various methodologies and models that can be implemented, the recommended methodology for Back Yard Adventures is to utilize the waterfall model. This was mainly selected due to several reasons. For one, requirements are clearly defined and static. However small enhancements can be accounted for if needed. The second is the size and complexity of the project. The project itself is not large and not complex, therefore this model will fit and integrate seamlessly with the overall development. The third reason is that this model is easy to manage, this of course is due to the overall clear structure and definable goals that are outlined when utilizing this model.

The end users of this system will be both the customers of Back Yard Adventures on the front end and the staff of Back Yard Adventures both the front and back end. Normally the waterfall model is not conducive with communication between users but due to the small size of the project, the staff of Back Yard Adventures will be involved at appropriate times throughout development. The project team will be located in house within the Back Yard Adventures staff. A programmer will be on boarded for development, the project team will include that individual and the rest of the staff at Back Yard Adventures. As mentioned before this project is relatively small, so overall development time and cost will be low.

## Conclusion

The proper utilization and selection of a software development methodology is imperative to the success of the project. Throughout the years, different methods and models have been created for nearly all projects. And as of late, with discretion, at times these models can be mixed and matched a bit to create a hybrid model if needed. For Back Yard Adventures, the model that will serve them best is the waterfall model. This is because the waterfall model encourages clearly defined requirements, appropriate for smaller sized projects, and easy to manage. With utilizing this project, the new system for Back Yard Adventure will be able to be fully created, tested, and deployed.

## References

Bugajenko, O. (n.d) *Agile vs. SDLC Development.* Study.com.

[https://study.com/academy/lesson/agile-vs-sdlc development.html#:~:text=Agile%20approaches%20prioritize%20flexibility%20and,on%20upfront%20planning%20and%20documentation.](https://study.com/academy/lesson/agile-vs-sdlc)

Schell, Deborah. (n.d) *The Agile Model in SDLC*. Study.com

<https://study.com/academy/lesson/the-agile-model-in-sdlc.html>

*SDLC - Waterfall Model*. Online Courses and eBooks Library. (n.d.). [https://www.tutorialspoint.com/sdlc/sdlc\_waterfall\_model.htm](SDLC%20-%20Waterfall%20Model.%20Online%20Courses%20and%20eBooks%20Library.%20(n.d.).%20https:/www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm)

Shikht, I. (2022, July 6). *SDLC - System Development Life Cycle*. Exoft. <https://exoft.net/system-development-life-cycle/>

Ransom, Noel (n.d) *Iterative Model in SDLC Methodology.* Study.com

<https://study.com/academy/lesson/iterative-model-in-sdlc-methodology.html>

Wamicha,Elizabeth (n.d) *V-Shaped Model in SDLC Methodology.* Study.com

<https://study.com/academy/lesson/v-shaped-model-in-sdlc-methodology.html>

*What is spiral model and how is it used?* (2019, August 15).

<https://www.techtarget.com/searchsoftwarequality/definition/spiral-model>

Zandergen, P (n.d) Systems Development Life Cycles: Software Development Process

<https://study.com/academy/lesson/systems-development-life-cycles-software-development-process.html>