CS 467 Capstone Project
Progress Report

The Importance of Analysis and Design in the Software Development Life Cycle

Carlos Osvaldo Saavedra Aguilar
Advisor: Sion Yoon
BS in Applied Computing
School of Technology & Computing (STC)
City University of Seattle (CityU)
saavedraaguilarcarl@cityuniversity.edu, yoonhee@cityu.edu

Abstract

Through the computing history, there have been situations where due to a bad planning and design in the Software Development Life Cycle, software products have ended in fatal consequences, from a radiotherapy machine that administrated up to 125 times the maximum amount of radiation that humans can deal with, to two airplanes crashing and killing more than 300 people, in both cases due to faulty software. To prevent situations like these to happen again, IT professionals must be aware that from the wide selection of tools that exist for planning and designing software, not all of them fit all projects, there are tools and techniques that due to their nature, might fit some projects better than others. The approach of this work is divided in two main sections, the first of them, comparing how recent graduates and professionals with little experience deal with this kind of decisions regarding what tools or development methodologies to use. The second, is conducting a research on methodologies, techniques, and listing some of the most common tools that are used nowadays for software development. At the end, this paper will serve as a guide for those who find themselves a bit confused regarding what methodologies and tools they should use, so software can be defect-free, preventing fatal consequences to happen.

Keywords: Software Development Life Cycle, Software Development Methodologies, Tools for Software Development, Software Project Management, Software Design, Systems Analysis.

1. INTRODUCTION

According to Roger Pressman (2010), the Software Development Life Cycle – SDLC – is composed by 5 general phases: communication, planning, modeling, building, deployment. While each of these plays an important role, the first three are fundamental for defining the basis of a successful software project. First of all, communication establishes the relationship between customer and client and sets the requirements for the project; if these requirements are unclear, problems will start to arise. Secondly, once requirements are unclear or misunderstood, planning will start from a corrupted base, and then, modeling will portray an inaccurate design. To this point, and according to Pressman's phases, 60% of the project would have happened, which would cause any flaw to be

complex to repair. Moreover, it would increment costs and delivering time, among many other consequences. According to Forbes, "unclear requirements, lack of coordination and planning, are some of the most common causes that lead software projects to fail" (Forbes, 2020).

Problem Statement

Paying little attention to planning software would result in an unprecise design, which will not meet the expected requirements. This may seem not as a big deal, but the problem becomes evident when nowadays, the software is capable of making almost its own decisions. Every day more and more people rely their lives on software products without having the minimum idea of who developed them and the attention and level of detail with which they were built.

José Gil states that fixing defects in the early phases can result up to 100 times cheaper than trying to fix them in advanced stages (Gil, 2016), but it is not just about costs. It is about being conscious that today great power has been given to software, allowing it either for great uses with the bests goals in mind, but giving it the ability to cause disasters, that in the worst scenario could result in the loss of innocent lives.

Said this, the problem is to understand the importance of the early phases in the SDLC – planning and design – not only to save costs but even to save lives. And more than that, it is not just about designing. It is about knowing what tools to use and what techniques and methodologies fit best to a given particular project. The goal is to inform about these tools, compare them, and explain some of their pros and cons and how software engineers and developers can take the most advantage of them.

Motivation

Throughout computing history, there have been cases where due to lack of planning and design in software projects, the systems produced have starred in catastrophic consequences. Back then, between 1985 and 1986, an x-ray and radiotherapy machine – Therac-25 – was found guilty of killing 3 people by administering up to 125 times the max amount of radiation the human body can handle. After a long investigation, Rodríguez (n,d.) mentions that it was determined the machine's software was not designed correctly; the software was not user-centered, which caused operators to administrate an overdose of radiation without them being aware of that.

In 1996, Ariane 5, a space rocket built by the European Space Agency, was destroyed by itself seconds after take-off. The final crash report, written by Louis Jacques (1996), showed that the flight control software had been taken from Ariane 4, the previous model, without considering the engines installed in Ariane 5 were way more powerful. The software was not modified to handle such immense power, which led the computer to a data overflow, crashing the system and then crashing – literally – the entire rocket.

More recently, in 2019, hundreds of airplanes – Boeing 737 MAX 8 – were grounded after two crashes that caused the loss of all 346 people on board. This new and technological plane relied on what Boeing calls 'MCAS', a software 'designed' to prevent the plane from stalling. In accordance to Gates & Baker (2019), the software was planned and designed as a low-scope secondary flight tool, which, unnoticed to the sight of engineers, became more and more important when being integrated with other flight systems, to the point where even the pilots themselves were prevented from taking control of the plane.

Approach

While some related documents and papers board this issue from a partial perspective, this paper is meant to integrate as much knowledge as possible and explain with enough detail as to make the reader capable of making the correct decisions regarding how to perform a successful software project, taking into consideration the main factors that could impact on the final outcome.

Conclusions

Having explained how important it is to perform a good analysis and design prior to every software project, the goal of this paper is to explain some of the principal software development methodologies, compare tools for planning, designing, and developing software projects, and how to perform a correct and efficient project management. At the end, the paper will serve as a guide to decide what tools and techniques fit best to a particular project depending on its needs and nature.

2. BACKGROUND

The process of developing software has always been a difficult task, and nowadays, where software applications become more and more complex, the difficulty in producing them also increases. Having detected this problem, engineers and programmers needed to come up with new methodologies and tools to simplify and enhance both the development of applications and the way in which software projects are carried out.

As software programs become smarter, the difficulty in planning projects and making estimations also increases since it results in complex establishing metrics for assets that are intangible. As a result of this, in the entire computing history – even in the recent days –, there have been software products that have failed due to lack of coordination, planning, erroneous design, and even misunderstood requirements, which eventually have ended up in catastrophic consequences, just as the discussed earlier.

It is important to address this problem to prevent more accidents due to defective software applications from happening in the future and to reduce potential risks as much as possible.

3. RELATED WORK

Software development has become such an important activity nowadays. There are applications, services, websites, among many other technologies, for almost every need, even for those most people did not know could exist. According to the U.S. Bureau of Labor Statistics (2021), the demand for software development professionals is expected to grow up to 22% from 2020 to 2030; a rate which they assure is "much faster than the average for all occupations" (U.S. Bureau of Labor Statistics, 2021). In a world where this activity shows such an impressive demand, it results very convenient to count on professionals who are aware of the importance of each phase in the Software Development Life Cycle – SDLC –, and how to correctly and effectively perform each of them. And, of course, all phases in the SDLC are meant to contribute in different and particular ways for the final result, but planning and design define the foundations for success, due to the fact that, is in these phases were the idea is born and the requirements are established, and if all of this is clear from the very beginning, it will be easier to set an accurate direction as well for the project as for the product, and will allow the following phases to flow without worrying about unexpected changes.

The final document will focus on software development methodologies, the activities, roles, pros and cons each of them has. Now, the success of a software project depends on the quality of tools and technologies used throughout the entire process; the way in which engineers and programmers take advantage of them also impacts the quality of the final product, and for this reason, these aspects will be covered too. Then, even having access to the latest or most advanced tools would be useless if the project is carried out under bad or wrong project management practices; communication, organization, teamwork, and coordination are key factors when determining not just quality for the product but even for the development process itself. Software development is a vast world, and it might result in overwhelming knowing where to start or which toolkit or methodology fits best the project needs, especially for newcomers to the profession.

Software Development Methodologies

Planning is the first step in any software development project – once requirements have been set, of course – and it may be one of the most challenging phases because if things are not clear from the beginning, the project will completely depend on uncertainty and unpredictability (Saini, Dubey, & Bharti, 2019). Now, the next step once everything is clear, would be to decide what development methodology fits best depending on the type and nature of the project. E-commerce and web development, business systems, mobile applications, open-source code; these are some of the various types of software systems that exist out there (Kendall & Kendall, 2011), and they all have different purposes and, therefore, different development needs. Agile – SCRUM, Extreme Programming, Agile Unified Process –, traditional, Personal Software Process; all of these are some of the different software development methodologies available, and just as with the different types of systems, they all offer different advantages (Pressman, Olguín, Brito, Quezada, & Castro, 2010). To this point, it might start to become a bit overwhelming deciding which to work with. Let's start by referring to agile methodologies; more than a specific methodology, they're a framework, where one of the major advantages is that they work best when collaborating with others, just as in a team project (Matthies, Huegle, Durschmid, & Teusner, 2019). They require great communication and organization skills, and for this reason, they should be used only when development teams have reached a mature point, enough to let everyone know what they're doing and to maintain motivation at all times (Dudhat & Abbasi, 2019). On the other hand, the traditional methodology – specifically, the waterfall model – is the development methodology that requires the least number of skills, and it might be a good option for newcomers to the software engineering world, but one of the major disadvantages of this model is the high sequential dependency along with all its phases (Casteren, 2017). This model should be used for a small project where

requirements will not change. The waterfall approach lacks the adaptability to unexpected changes in the project life cycle.

Tools for Software Development

If there are many methodologies for software development, there are a thousand more tools for the same purpose. Of course, this is great because that means there is a tool for every need, and that will, most likely, be incredibly useful for a particular task. Once more, the problem comes into play when deciding which one to use. One of the first things that come to mind is to have a version control system for tracking changes in the project; and Git might be the best tool for this purpose. It is a great tool and is worth knowing how to effectively use it; some of the most common errors and bad practices when using this tool happens when applying Git Workflows, creating commits, pushing commits, merging, fetching, branching, and reverting (Eraslan et al., 2020). Now, it would be convenient to decide what tools to use depending on the development methodology, for the Aspect Oriented Programming – an emerging paradigm – or for the well-known Object-Oriented Programming paradigm, UML diagrams result in the best design tool for modeling classes (Babu & Vijayalakshmi, 2008), while for the traditional waterfall model it would be better to use flow diagrams. Then it's time to start coding, and for doing so, an IDE is required, and again, there are hundreds of options to choose from. Some factors on deciding what IDE to use include: if the programmer needs highlight syntax, IDE speed, in-product compilation, built-in debugging, class browsers, and so on (Cogswell, 2015). Just after – or along – with the coding phase, there comes the testing phase, and once more, the trouble of deciding which tool to use comes into play, and to find the right tool, first it is important to know what kind of testing is needed: stress, load, regression, functional, unit, performance, acceptance, security, or open source (Musa, Al-Qutaish, & Muhairat, 2009).

Software Project Management

To find the development methodology and development tools that best fit the project is great progress, but things do not stop there. It would be useless to have the greatest approach or the greatest and most expensive tools if the team is unable to take the most from them. Software projects are influenced by various problems and factors that can result in ineffective management (Laktionov & Stratiienko, 2021). Project Management plays an incredibly important part in any software project since it must be present in every phase, from planning and estimating to analyzing risk factors, leading and directing the project, or even representing it (Fairley, 2003). Correct and effective project management brings many benefits; first of all, it ensures that customers objectives and stakeholder's expectations are met, it increases chances of success, helps to respond to risk in a timely and efficient manner, optimizes working resources, and manages constraints (Project Management Institute, 2017). For this to be possible, the development team, along with the project manager, can make use of some tools, such as Microsoft Project; which is a tool that allows to list tasks, define deadlines, and generate different types of diagrams, JIRA; a tool which results best when used in agile methodologies, or GlassCubes; a hosted CRM solution that eliminates the need for deploying clients and backend software and therefore, no complex configurations or installations are required (Mishra & Mishra, 2013).

4. APPROACH

Documents, papers, and essays evaluated in the related work section board single topics. It is a bit difficult to find documentation where more than just a few subtopics are covered. Generally, the integration of them can be found just in books with hundreds of pages, but results hard to find a 'quick reference guide' and sometimes not everyone may be interested in reading all the theories that books tend to contain, and due to the increasing need of developing software projects in short periods of time, it is convenient to have access to a source which covers as much as possible while focusing just on the important information, so programmers and IT people can spend more time on what really matters – their projects – and less time trying to look for documentation. According to Garret Hollander (2020), in 2012, Gartner conducted an assessment where they found that it takes an average of 18 minutes to find each of the documents that really meet the researchers' expectations, and doing basic math, it would mean as much as 3 hours to find just 10 documents or papers that really contain what is needed. It is still good, of course, to have access to books with lots of information, or to papers which focus just on a single thing and dive deep into it, but for some people, it may result more convenient to go straight to the point, especially when working with little time or when deadlines start to get closer. Having both types of sources bring different advantages; on the one hand, if people have enough time, they can go deeper into theory. On the other hand, quick guides like this save time and go straight to the point, covering

as much as possible on different subtopics for a specific field, in this case, software engineering.

For this concentration of useful information to be possible, the sources this paper references have been peer-reviewed and come from trustful sources, such as the ACM or IEEE repositories, PMBOK, Software Engineering books, and so on. A detailed review of each of the sources has been conducted in order to find the most important points that are worth integrating into this project, so future readers can be confident that the information shown here comes from reliable authors specialized in each of the topics.

In addition to this paper, a small project will be developed to show in a 'hands-on-practice' way some of the methodologies, planning and design tools, and project management practices presented and explained within the document. Said project will be no more than a graphical calculator, which will be capable of accepting mathematical equations entered by the user and drawing their correspondent graphic on a Cartesian plane. First, a Software Requirements Specification document will be written to state the exact functional and non-functional requirements for the application. For the implementation part, Java will be selected as the coding language, therefore, the design will rely on UML diagrams. For managing the project and the tasks to be completed, Microsoft Project will be used, and finally, Java Test Classes will be developed to test the final application and validate that it actually meets the proposed initial requirements. To this point, it is still unclear what kind of methodology will be used, but since it will be a project developed just by one single person – me –, the Personal Software Process might be the methodology that best fits to it.

5. DATA COLLECTION

This project is more about gathering information from already existing related work rather than developing or discovering new things. Despite that, a little questionnaire of 10 questions has been developed to conduct a study. This questionnaire will be given to 20 students of IT-related careers – software engineering, computing systems – in order to discover how they are feeling regarding their knowledge in software development methodologies, tools for developing software, and project management. It is worth noting that all students are on their last semester, meaning that by this point, they must have acquired enough knowledge to have a wide and clear understanding of what they will be asked. In addition to the 20 students, the exact same questionnaire will be applied to 20 people who are already working in a professional environment – regarding IT, of course – and have between 1 and 2 years of experience. In total, 40 people will be answering the questionnaire in order to contrast how students and professionals perceive the knowledge they have acquired along with their careers and working experience. The purpose of this questionnaire is not to dive deep into complex questions. Instead, it is about collecting their perceptions regarding the topics of this paper in order to prove that, indeed, IT people might, at a certain point, find themselves confused between all the methodologies and development tools that exist out there. The questionnaire will be developed using Google Forms, and a sample of it can be found at the end of this document in the appendices section.

As for the collection of information from already existing documents and papers, this will be done through trustful repositories and sources, just as previously mentioned in the approach section.

6. DATA ANALYSIS

The questionnaire was provided to all 40 people, 20 students and 20 professionals. The data collected from the students is as follows:

- Of the 4 methodologies mentioned (waterfall, agile, PSP, spiral), 10 students have stated that they have heard about them at least once. 7 students have heard only of waterfall and agile, 2 have heard of PSP, and just 1 has heard of waterfall and spiral.
- 11 students know the pros and cons of each of the methodologies they've heard about, while 9 do not.
- 11 students know the kind of projects each of the methodologies is suitable for, while 9 are not sure.
- 13 students would not feel sure about deciding what methodology to use if they were in charge of a software project. 7 students said they would.
- 11 students have been involved in software projects that have faced timing or budget issues, while 9 have not.
- Again, 11 students would not feel confident in choosing the right development tools that their projects need. 9 said they do feel confident about that.
- 14 students think the project's final outcome is directly impacted by the selection of the right development tools. 6 students think it does not impact the result.
- Finally, 17 students are sure that a good project management leads to a high

quality product. 3 other students do not share this same posture.

Now, the results obtained from the 20 professionals are as follows:

- 13 professionals have heard of all the 4 methodologies asked in the questionnaire. 5 have heard of the waterfall and agile methodologies, and 2 have heard of waterfall, agile, and PSP.
- 13 professionals know the pros and cons of each of the methodologies they've heard about, while 7 do not.
- 13 know the kind of projects each of the methodologies is suitable for, while 7 are not sure.
- 10 professionals would feel sure about deciding what methodology to use if they were in charge of a software project. The other 10 said they would not.
- 11 professionals have been involved in software projects that have faced timing or budget issues, while 9 have not yet.
- 12 professionals would feel confident in choosing the right development tools that their projects need. 8 said they would not feel confident about that.
- 17 professionals think the project's final outcome is directly impacted by the selection of the right development tools, while 3 think it does not impact the result.
- Finally, 19 professionals are sure that a good project management leads to a high quality product. 1 single other professional do not share this same posture.

The data shows that professionals have, indeed, a little more knowledge regarding these topics, although the gap between the results shown by the students is not really far from the ones shown by the professionals. Knowing this, three conclusions can be deducted from this:

1. Experience brings knowledge, therefore, it allows professionals to have a wider and clearer understanding.
2. Students that are in their last semester have reached a mature amount of knowledge, and although most of them do not feel confident enough to make important decisions when working on a software project yet, this is something that time and experience will improve.
3. Although just a few, there are still people who, unfortunately, do not see the importance of project management and how it impacts the product's quality.

7. SOFTWARE DEVELOPMENT METHODOLOGIES

According to the Oxford English Dictionary (n. d.), a methodology is "a set of methods and principles used to perform a particular activity." In other words, it is a set of defined steps to achieve a specific goal in a specific way and following determined rules. A software development methodology is exactly the same. It is a framework that allows engineers, programmers, and developers to perform a software development project under certain and specific rules and steps. Among the most common software development methodologies, there are the agile methodology, the traditional methodology – also known as the waterfall approach –, the spiral methodology, and the Personal Software Process. Naturally, each of them is different, providing different pros and cons, and allowing the development team – or software engineer in charge of the project – to choose one that best adapts to the project needs. Not all projects have the same requirements. Some are clear from the very beginning, and others cannot be defined and need to be constantly changing, sometimes the users and stakeholders are involved, while some other times they are not, some methodologies work best with small projects, while others are suitable for large and complex ones.

7.1 Traditional Methodology

Also known as the Waterfall Approach, for a long time, this was maybe the most famous – and only – methodology, especially in the early days of software development. Back in those days, software was very different from what it is now. There were not as many programming languages and technologies as today. Nowadays, this methodology is in its last days, and although it is easy to apply, it falls short of today's ways of producing software.

According to Pressman (2010), the waterfall methodology is formed of 5 simple steps:

1. **Communication:** this is the very beginning of the project. This step is where the project lead meets the client, and all involved stakeholders and the requirements are established.
2. **Planning:** in this step, all estimations are made (time, budget, size, resources required), all tasks to do during the entire project are established and organized.
3. **Modeling:** in the third step, the analysis and design of the system take place, all components are designed, diagrams are developed, and all considerations are – or at least, should be – taken.

4. **Construction:** the fourth step is, maybe, the most exciting phase for all software enthusiasts. Here is where the coding and testing take place – yes, coding and testing in the same step! –.
5. **Deployment:** finally, once the coding and testing phase is finished, it is time to put the system into production and deploy it to the real world.

As can be seen, the waterfall methodology is a very systematic approach, all steps are tied to each other, and there, the dependence among them is very high. Another important thing to note, and maybe the worst drawback of this methodology, is that once a phase has been completed, the development team is unable to go back. So, imagine a defect was introduced in the modeling phase and it was detected until the deployment step, that simply would mean troubles, as in order to solve it, the development team would need to return as far as to the phase where the defect was injected. Now, imagine this defect was found on a crucial component that interacts with many other components in the system. That would mean that the entire code needs to be modified because, of course, the coding was based on the defective design, and therefore, the test cases were too! Believe it or not, this happens more frequently than you imagine. As humans, that design and code, errors are awaiting for the slightest distraction to come into play, and when working with complex systems, it is easy to make mistakes. By this point, not just the work that had been done is unuseful. The application needs to be fixed, which of course, consumes more resources than the ones expected, leading, in most cases, to overtime and over budget.

This does not mean that the waterfall methodology is bad. After all, it is simple to use and easy to understand, and its clearly segmented structure makes it easy to know when a step has been completed or not. The traditional approach is good to be used in the following scenarios:

1. When working with small projects and small teams.
2. When the project's requirements have been clearly stated and understood from the very beginning, and no one in the development team has questions about what is expected from the application to be developed.

On the other hand, it is not recommended when:

1. Requirements are ambiguous or change constantly.
2. The client wants to see the application as soon as possible. Remember that the coding is the fourth phase, so it means the client will be wondering how their project is looking for a long time.

7.2 Agile Methodology

In reality, saying an 'agile methodology' does not tell anyone the methodology that is being used. While the traditional methodology is just the waterfall approach, in the agile methodologies, there are several frameworks – SCRUM, Kanban, and so on –. They all share the same principle. The only difference is the steps needed to achieve the objectives. For example, both SCRUM and Kanban have 5 general steps, but they differ slightly from each other, and despite that, they both are agile methodologies since they are based on the same principles and follow the same general rules.

Said this, the rest of this section will be focused on SCRUM, which is, maybe, the methodology – and framework – that is being used the most nowadays.

As said before, all agile methodologies have things in common, and they all start from the same place: The Manifesto for Agile Software Development. This is a document that establishes what is needed in order to develop in an 'agile' way. Among the 12 principles the manifesto contents, the ones that draw the attention the most are:

- "Welcome changing requirements, even late in development."
- "Deliver working software frequently."
- "The best architectures, requirements, and designs emerge from self-organizing teams." (Beck, K., Beedle, M., et al, 2001).

Now, back to SCRUM, one of the things that make SCRUM be what it is, is the sprints. According to Pressman (2010), a sprint is a "unit of work needed to complete a requirement of the project, that must adapt to a defined amount of time." In simple words, a sprint is a period of time in which the development team and all involved participants focus on certain specific tasks. Having understood what a sprint is, the steps that compose the SCRUM methodology – framework – are:

1. **Product Backlog creation:** once the requirements have been collected, the product backlog creation consists of transforming those requirements into "user stories", which is the way that the SCRUM team uses to visualize and understand the project's requirements in an easy way. The final compilation of

requirements transformed into user stories is known as the product backlog.

2. **Sprint planning:** this is where the sprint is planned. The time and activities that will be done in the sprint are set. Additionally, the resources that each task – or user story – will need are defined.
3. **Work on sprint:** just as the name says, this is about working on the tasks defined in the previous step and making sure to complete each of them. As the tasks are completed, they are taken out of the work "to do", so this way, the team can measure the sprint's progress.
4. **Test and product demo:** every time a sprint ends, the tasks and components developed in it need to go through a testing phase to ensure that they are complying with the client requirements.
5. **Retrospective:** at the end of each sprint, the entire team meets to discuss how things went, what worked well, and what can be improved for the next sprint. After this, the team starts planning the next sprint, and the process repeats until the project is finished and all sprints are complete.

As can be seen, SCRUM – and agile methodologies in general – work iteratively by increasing the progress of the project by each sprint.

Another important thing in SCRUM is the daily meetings, which should last no more than 15 to 20 minutes, where the team discusses how they are doing and whether they find some issues or not.

SCRUM and agile methodologies are recommended to use if:

1. The team knows how to effectively communicate. Communication is crucial for the success of the methodology.
2. The project requirements may be changing from one moment to another.
3. The client wants to see progress. Since each sprint delivers a "tangible" outcome, the client and stakeholders can keep track on how the project is going.

On the other hand, SCRUM methodologies should be avoided if:

1. The team does not know how to accurately communicate among them.

2. There is no one in the team who knows or has experience working with these kinds of methodologies. This would only cause confusion and make the methodology seem like is impossible to understand.

7.3 Spiral Methodology

The Spiral Methodology is an evolutionary development process, actually pretty similar to the incremental models – such as SCRUM –, but one of the differences between them is that here, sprints are not required. One of the main objectives of this methodology is to reduce the risks of the projects by developing them into cycles, where little parts of it are produced. Small tasks allow to have better control, also, after each cycle, a planning phase occurs for the next one, and this repeats until the project ends.

The phases that compose this methodology are:

1. **Communication**
2. **Planning**
3. **Modeling**
4. **Construction**
5. **Deployment**

At this point, it can be seen that the phases are exactly the same as the ones used by the waterfall methodology – this is why it is not worth explaining them again, remember that the goal of this paper is to go as straight as possible –, but the difference is that here, they are not sequential. Well, they are; they happen after the previous is finished, but the point is that the entire project is divided into small deliveries, and for all of those deliveries, all the phases must be carried out, making the project flow in a cyclical way – spiral –. For example, the first delivery might be the design alone, and for that task, all 5 phases must be put into practice. Then, the second task could be to deliver a module of the user interface, and for that task, all 5 phases must happen again. This is why the methodology is called spiral because, after each small delivery, the team goes back to communicating, planning, modeling, building, and deploying the next task. Remember that in the waterfall approach, the entire system is done at once, and these phases are just boarded once, making a unique and final delivery, with all the risks it represents, of course.

The spiral methodology should be used when:

1. The client wants to see how the project is taking shape and wants to manipulate functional – although small – parts of the project as soon as possible.
2. The project's deadline is not that much of an issue; constant planning and design require plenty of time.

The spiral methodology is not recommended when:

1. The project is facing time constraints or issues.
2. The project's budget is limited. Just as with time, constant planning and design consume resources.

7.4 Personal Software Process

The Personal Software Process – PSP – is a methodology that, as its name says, is meant to be put in practice when there is only one individual working on the project. For this same reason, this methodology is not that popular since most projects need entire teams of people working in them rather than just only one. Despite this, it is worth knowing that this methodology offers some advantages and gives each developer the opportunity to know and understand his efficiency and many other factors when working on a personal project. This way, it will be easier to make estimations when the time to be part of a team comes into play.

The PSP is divided into different levels that go from the basics, such as understanding how things are happening before the use of the methodology is put into practice – the baseline –, to complex things, such as developing graphs to compare the performance the developer acquires along the project he works on. This methodology provides different templates and scripts for each of the levels in order to guide the developer through all the things that are necessary to know and understand his efficiency and performance. Some of these scripts include planning, design, development, testing, and so on. The levels that compose the PSP are:

- **Level 0:** establishes the baseline. This will be useful later to compare how the developer performs before and after using the methodology.
- **Level 0.1:** level 0, plus, requires the developer to set personal coding standards and a way to measure the code that is produced. Additionally, suggests the developer to list Personal Improvement Proposals – PIP's –.
- **Level 1.0:** level 0.1, plus, includes planning and testing reports for each program the developer works on. Also, requires resources estimations.
- **Level 1.1:** level 1.0, plus, requires planning tasks and schedules for each development project.
- **Level 2.0:** level 1.1, plus, design and code reviews are integrated into the methodology.
- **Level 2.1:** level 2.0, plus, a design specification and analysis techniques must be established.

Previously, there existed level 3 in the methodology, but now, instead of it, the TSP – Team Software Process – is added. TSP would be like the 'last level of PSP', although, as its name says, it is focused on teams. It is supposed that developers who have come this far have acquired enough personal knowledge to migrate to a team development environment.

These PSP levels are meant to incrementally improve the personal development process and is suggested to move forward from one level to another when the developer has understood and put in practice what each of the levels suggests.

Just as the rest of the methodologies, PSP also provides a list of phases that must be carried out for every project. The phases are:

1. **Planning:** requirements are established, and estimations are made, including the number and type of defects that are estimated to be injected during the following phases. This estimation of defects will later be useful to help the developer understand the phases where he is the most vulnerable to make mistakes.
2. **High-level design:** each component of the system must be designed. The integration of all of them also needs to be designed.
3. **Design review:** different from any other methodology explained in this paper, the PSP establishes a phase specifically for design review. At this point, it can be seen that PSP cares that much for the design of the system. After all, it is crucial when defining the success of both the application and the project.
4. **Development:** 'the interesting part', where design is transformed to code.
5. **Post-mortem:** just like in SCRUM, at the end of the project, the methodology requires the developer to make an analysis of the project itself and to identify what went well and what can be improved. Here, the estimation of defects made in the planning phase comes into play.

The PSP is a methodology easy to understand. It is systematic and provides scripts and templates, enabling the developer to go step by step without missing anything. Therefore, this methodology is recommended when:

1. Working alone in a small software project – that's why it is called Personal Software Process –.
2. The developer has little experience and wants to: know himself more regarding the development of software projects, know how he performs, what he does well, and what areas of opportunity he has.

On the other side, this methodology should be avoided if:

1. There's little time to develop the project. Remember, this is a systematic and step-by-step methodology, so it consumes lots of time.
2. For experienced developers who have been working on software projects for a long time, this methodology might be tedious as it is more 'beginner' focused.

8. SOFTWARE DESIGN AND DEVELOPMENT TOOLS

8.1 Planning and design tools

UML Diagrams: diagrams meant to be used for modeling Object Oriented applications. Some of the diagrams UML offers are class diagram, activity diagram, state diagram, and so on.

Data Flow Diagrams: these diagrams describe the flow that a software application is meant to follow. A data flow diagram specifies inputs, processes, and outputs, in addition to illustrating conditional paths and more.

Input Process Output Charts: these might be the most basic designing tools, also known as IPO charts. What they do is organize all the inputs that the program needs, then specify how those inputs will be processed, and finally, what the expected output will be.

Algorithms: another basic design tool, but fundamental in any software project. Algorithms establish the general flow of the application and establish the steps needed from the very beginning to the end of the program execution.

Pseudo-Code: pseudo-code consists of writing, line by line, all the instructions of a software program, but instead of writing them using any coding language, they are written using natural human language, in order to allow the programmer to understand with daily words what the program will do.

8.2 Development tools

Coding Languages: coding languages might seem something obvious when developing software. Just as with methodologies, not all coding languages fit all needs. Nowadays, Object Oriented languages are very popular, being Java and Python the most used ones. Structural languages, like C, still being the base of legacy systems, and one or other current systems might still be developed with it.

IDE: choosing a good Integrated Development Environment saves time, allowing the developer to make use of the tools it might offer. Most IDE's offer coding snippets, real-time syntax check, and debugger. Some others sacrifice these kinds of tools in pro of speed and small size in disk. For Object Oriented languages, Java, specifically, the most popular IDE's are NetBeans – maybe the most known one –, Eclipse – great for developing mobile apps –, IntelliJ Idea – a very smart IDE which offers lots of tools –, and BlueJ – a graphical IDE great for beginners –.

Testing tools: Testing tools are also important when developing a software project. Jenkins, for example, is an open-source automation server that allows developers to test and deploy their applications, making use of the hundreds of plugins it has to offer. Appium, a great tool for testing native, hybrid, and web mobile applications. Selenium, a tool that automates browsers and allows developers to automatically test web applications. On the other hand, local testing tools might include Java Testing Classes, ideal when testing needs are very specific.

9. SOFTWARE PROJECT MANAGEMENT

The correct management of any project is crucial for determining its success, and it is not the exception with software projects. Actually, these kinds of projects might be the ones that most need correct and effective management. One of the biggest problems of software is that many people find it difficult to establish a global way of measuring something that is intangible, just as software, so many techniques have been proposed to achieve this. In the past, software projects were commonly measured by the number of code lines that were needed to be developed, and the cost of an application was established by the number of code lines too. However, although this technique is still being used by some project leads, one of the techniques that nowadays is used the most is measuring software programs by the 'function points' they have. In other words, the trend now is to measure software by the number of functionalities they provide, and the estimations of cost and time needed to develop them are directly related to these function points. This way of measuring software is known as COSMIC Sizing.

Some tools to help manage software projects are:
GDPA: a group of tools developed by Bremen University in Germany, which provide many functions to model and manage processes, ideal for the 'V' methodology.

ProVision BPMx: helps define the process and automate the workflow.

Microsoft Project: a tool part of the Microsoft Office suite, great for developing many types of diagrams: Gantt chart, fishbone diagram, network diagram, and many others. This tool helps organize tasks and assign resources to each of them.

JIRA: a tool ideal when working with agile methodologies, provides a pluggable integration framework.

GlassCubes: is a CRM-hosted solution, which eliminates the need to deploy clients and backed software.

10. FINDINGS

Software development is a vast world, and the many technologies that exist out there to help engineers and programmers achieve their goals might seem overwhelming when someone has little experience. The questionnaire applied to students and professionals revealed some interesting things:

- Experience makes the difference, although students seem to be well prepared for the real world.
- In the beginning, the theory is important, but later this is replaced by experience.
- Having a paper that integrates different development methodologies and explains them briefly and with enough information helps IT people to save time when looking for something for their projects.
- Explain the main planning, design, and development tools that are worth understanding which one fits best the needs of a particular project.
- Project management is crucial to determine the project's success. Good project management, plus the use of the right methodology and tools, equals a successful project and a quality product.

11. FUTURE WORK

Technology evolves quickly, and techniques, tools, and methodologies that now are new and unique, could just in a few years – or in some cases even months – be replaced by new disruptive things, and this way, it becomes a cycle. The methodologies and tools shown within this document are either the trend, the most common, or the newest ones at the time of its writing. The goal as future work is to constantly update this document, in a way that even when time comes by, the contents of it are up to date to ensure the paper is still worth reading. Maybe update it every year, or accordingly to what the industry has to offer, who knows, maybe even just after a few weeks the paper is finished, new tools could arise…

12. CONCLUSIONS

Software development is by itself a complex task. It requires attention to details, logic, a lot of thinking, and being able to see things in a different perspective. However, it is rewarding to see how after all the hard work, solutions are provided, that in most cases, simplify and improve processes, automate tasks, and make life easier for people working almost in every industry. Now, in order to make the IT and software development industry easier itself, tools and methodologies have been developed throughout time, but it is important to know how to choose the correct things to use, the ones that will actually bring benefits for the development team. Not all methodologies are suitable for all projects, and exactly the same happens with planning, design, and development tools; each of them provides different benefits, and fit some projects better than others. Knowing how to make the right choice might be a complex task, especially when people do not have that much experience, and some times looking for the right tools and techniques to use could mean a lot of research time, that otherwise, could be used to focus on their projects. Not just knowing how to choose the right things brings benefits, it also improves and ensures quality in the final product, that otherwise, could be full of defects and in the worst case, fail when needed the most, bringing fatal consequences just as in the cases shown in the introduction section. There might be thousands of tools for software development, and it may be difficult to know which one to choose, some times the best one could even be unknown for the development team, so it is a good practice to be well informed before making any kind of selection.

13. REFERENCES

Forbes. (2020). Council post: 14 common reasons software projects fail (and how to avoid them). Forbes. https://www.forbes.com/sites/forbestechcouncil/2020/03/31/14-common-reasons-software-projects-fail-and-how-to-avoid-them/?sh=2a83b703798c.

Gil, J. M. (2016). El problema malévolo y su coste en el diseño de software. Blog Yunbit Software. https://www.yunbitsoftware.com/blog/2016/10/07/problema-malevolo-y-su-coste-en-diseno-de-software/.

Gates, D., & Baker, M. (2019). The inside story of MCAS: How Boeing's 737 Max system gained power and Lost safeguards. The Seattle Times. https://www.seattletimes.com/seattle-news/times-watchdog/the-inside-story-of-mcas-how-boeings-737-max-system-gained-power-and-lost-safeguards/.

Jacques, L. (1996). Ariane 5 Flight 501 Failure Report. ARIANE 5 failure - full report. http://sunnyday.mit.edu/nasa-class/Ariane5-report.html.

Rodríguez, C. (n, d). Accidentes Therac-25. https://lsi2.ugr.es/mvega/docis/aluwork/roddesastres/therac.htm.

U.S. Bureau of Labor Statistics. (2021). Software developers, Quality Assurance Analysts, and testers: Occupational outlook handbook. Retrieved October 20, 2021, from https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm.

Matthies, C., Huegle, J., Durschmid, T., & Teusner, R. (2019). Attitudes, beliefs, and Development Data Concerning Agile Software Development Practices. 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET). doi:10.1109/icse-seet.2019.00025

Dudhat, A., & Abbasi, M. A. (2019). Discussion of Agile Software Development Methodology and its Relevance to Software Engineering. doi:10.34306/ajri.v3i1.536

Casteren, W. V. (2017). The Waterfall Model and Agile Methodologies: A comparison by project characteristics. doi:10.13140/RG.2.2.10021.50403

Saini, G. S., Dubey, S. K., & Bharti, S. K. (2019). Novel algorithm for software planning & development. Proceedings of the Third International Conference on Advanced Informatics for Computing Research - ICAICR '19. doi:10.1145/3339311.3339315

Eraslan, S., Ríos, J. C., Kopec-Harding, K., Embury, S. M., Jay, C., Page, C., Haines, R. (2020). Errors and poor practices of software engineering students in using git. Proceedings of the 4th Conference on Computing Education Practice 2020. doi:10.1145/3372356.3372364

Babu, C., & Vijayalakshmi, R. (2008). Metrics-based design selection tool for Aspect Oriented Software Development. ACM SIGSOFT Software Engineering Notes, 33(5), 1-10. doi:10.1145/1402521.1402522

Cogswell, J. (2015). Choosing an IDE that's right for you. Retrieved October 22, 2021, from https://insights.dice.com/2015/05/19/choosing-an-ide-right-for-you/

Musa, K. M., Al-Qutaish, R. E., & Muhairat, M. I. (2009). Classification of software testing tools based on the software testing methods. 2009 Second International Conference on Computer and Electrical Engineering. doi:10.1109/iccee.2009.9

Fairley, R. E. (2003). Software project management. doi: https://dl-acm-org.proxy.cityu.edu/doi/pdf/10.5555/1074100.1074810

Laktionov, S. Y., & Stratiienko, N. K. (2021). Specifics of software project management in IT. ISSN: 2222-2944

Project Management Institute. (2017). A guide to the Project Management Body of Knowledge. Newtown Square, Pennsylvania: Project Management Institute.

Mishra, A., & Mishra, D. (2013). Software project management tools. ACM SIGSOFT Software Engineering Notes, 38(3), 1-4. doi:10.1145/2464526.2464537

Kendall, K. E., & Kendall, J. E. (2011). Análisis y Diseño de Sistemas. México, New Jersey: Prentice Hall.

Pressman, R. S., Olguín Campos Víctor, Brito Enríquez Javier, Quezada, V. C., & Jorge Ferro Castro Bárbaro. (2010). Ingeniería del software: Un enfoque práctico. McGraw-Hill.

Hollander, G. (2020). How long does it actually take to find a document? Intelligent Information Management Resources. Retrieved November 5, 2021, from https://resources.m-files.com/blog/how-long-does-it-actually-take-to-find-a-

document-dissecting-the-many-stats-out-there.

Oxford English Dictionary. (n. d.). Methodology. methodology noun - Definition, pictures, pronunciation and usage notes | Oxford Advanced Learner's Dictionary at OxfordLearnersDictionaries.com. Retrieved November 18, 2021, from https://www.oxfordlearnersdictionaries.com/us/definition/english/methodology?q=methodology.

Beck, K., Beedle, M., et al. (2001). Manifesto for Agile Software Development. Retrieved November 19, 2021, from https://agilemanifesto.org/.

COSMIC. (2020). The open standard for software size measurement. Cosmic Sizing. Retrieved December 4, 2021, from https://cosmic-sizing.org/.

APPENDICES

Appendix A

1. Mark the software development methodologies you've heard of. (Agile, Waterfall, PSP, Spiral)
2. Are you aware of the pros and cons of each of them?
3. Do you know the kind of projects they work best with?
4. Imagine you're in charge of selecting the development methodology for a software project for an entire development team. Would you feel confident enough as to make the right choice considering the particular needs of the project?
5. How do you plan software projects that you work on?
6. Have you ever worked on a software project that had issues regarding timing or budget?
7. What would you say was the reason (or reasons) that led it to face those problems?
8. Would you feel confident enough as to choose the development tools (version control, testing, IDE, etc.) that best adapt to the needs of a specific software project you're working on?
9. Do you think choosing the right development methodology and tools impact the project's final outcome?
10. Do you think a weak project management could impact on the project's final outcome even if you're working with the correct development methodology and tools?

Appendix B

Link to YouTube video, showing the application demo: https://www.youtube.com/watch?v=jnzlNDj61Yo

Link to YouTube, showing the voiceover presentation: https://www.youtube.com/watch?v=FXw72zwjkL0

Link to GitHub to the source code of the application (and some diagrams): https://github.com/PianissiMan/CS497-Project