MIEIC – October 2017

Software Development Methods

ESOF

Class 1 - Group F

Diogo Peixoto Pereira – up201504326
Maria Eduarda Santos Cunha – up201506524
Pedro Miguel Ferraz Nogueira da Silva – up201505460

Index

1. Iterative Software Process

Iterative Software Process is a software development technique that, although it first came around in 1950, was only formally presented in 1970, in the article “Managing the development of large software systems”, by Winston W. Royce.

It was created to respond to inefficiencies found in the Waterfall model.

It consists of a cyclic process of prototyping, testing, analysing and refining a product. Essentially, changes based on results of testing the latest iteration are made to the project, improving quality and functionality with each iteration.

Although changes are applied all throughout the process, they are cheaper and easier to implement in the earlier stages of development.

The Iterative Development’s **principles** and **practices** are to:

1. Manage the requirements instead of the tasks, based on use cases and non-functional requirements;
2. Meet business goals, dates and budgets. It is preferable to change the requirements to fit these than the other way around;
3. Do only what is necessary. For every requirement there should be a user goal. And analyse what is already implemented as to make sure they are meeting the business goals;
4. Adapt to changes or add-ons after the development begins;
5. Always begin with a basic implementation of some requirements to demonstrate the key aspects of the system;
6. Design should be around isolated and easily found modules of related requirements. Per iteration, one module should be completed or reviewed;
7. Work is to be done in short cycles;
8. While working on an iteration, the customer cannot change the requirements for that iteration, but the development team may change them by letting go of features in order to meet deadlines;
9. Modifications must become easier to apply as iterations progress. Otherwise, redesign is needed.

The Iterative Method comprehends the following **phases** (fig.1):

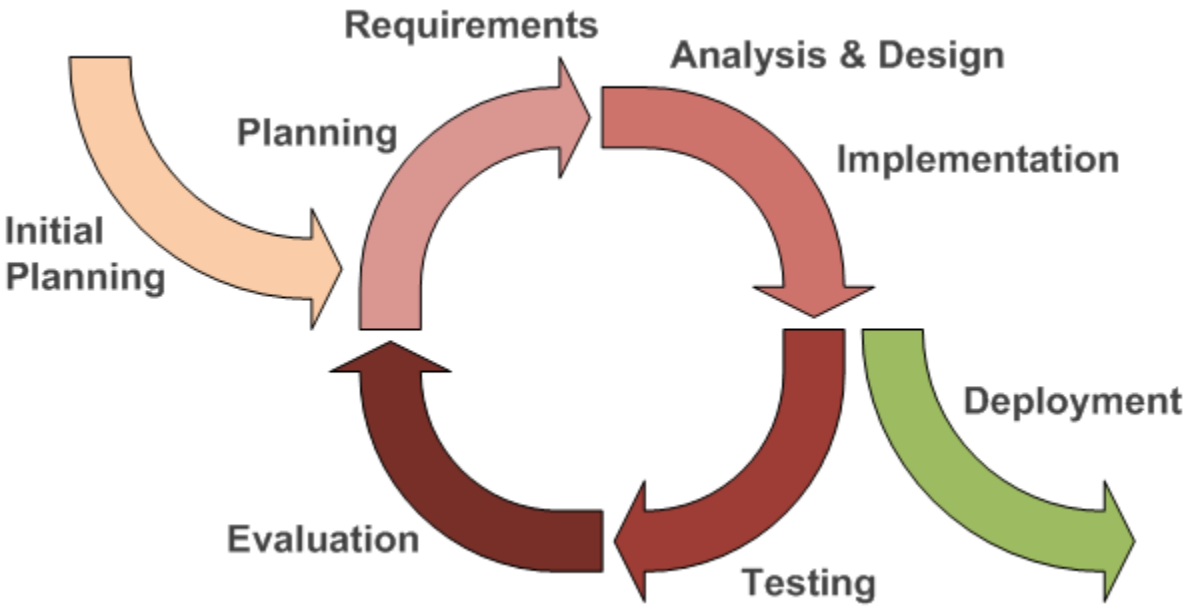

Figure 1: Iterative Method Phases

This process has proved itself **very useful** as, when properly applied, it ensures a product is the best possible solution. That would be because it gets the customer involved, as he is required to give feedback every few iterations. This way, it is sure the product will meet the user's needs.

Also, if applied from an early stage, it allows for significant cost savings as serious misunderstandings are made evident early on and it's easier to respond to them without creating ripple effects or inconsistencies in the project.

While initially it may seem like a morous process, it is important to notice that each iteration will be faster as less and less is required to be corrected or added.

On the other hand, phases of an iteration don't overlap as they are rigid and if all requirements are not gathered up front for the entire lifecycle, costly system architecture or design issues may arise.

Due to this, iterative development would be **safer to use** when all requirements of a complete system are clearly defined and understood. It is quite useful if dealing with a big project. However, generally speaking, it is **not advisable to use** for front end development. Furthermore, it demands a higher level of technical expertise, discipline and buyout from

the team, since it often requires that **most team members are apt to perform several types of task**.

Some examples of **successful projects** using this method are:

- In the mid-70s, when iteration was on its early days, there was a quite successful Iterative project for the USA Navy helicopter-ship system LAMPS (Light Airborne Multipurpose System). It was "a 4-year 200 person-year effort involving millions of line of code, it was incrementally delivered in 45 timeboxed iterations (1 month per iteration)".
- Another early and striking, but less known, example is the heart of the NASA Space Shuttle software: the "Primary Avionics Software System". This was built from 1977 to 1980, "applying Iterative and Incremental Development in a series of 17 iterations over 31 months, averaging around 8 weeks per iteration".

By using the Iterative Software Development method, the following artifacts are produced:

During Analysis…

Although all the physical work done in this phase tends to be discarded, several outputs survive, including:

- Vision statement;
- High-level description of the project's objectives, use cases and actors;
- Preliminary project plan;
- High-level software architecture diagram;
- Business case for the project.

During Design and Development …

- Detailed use cases describing the majority of the system's behaviour;
- Class and sequence diagram describing attributes, methods and all the relationships between objects;
- Detailed system architecture diagram;
- Software code;
- Published API;
- Detailed project plan describing subsequent iterations;
- Preliminary test plan to verify the operation of the software;
- Sometimes, there may be also a preliminary user manual.

During Deployment…

- **Deployment Plan:** finalized Deployment Plan which is used as the roadmap for delivery to customers;
- **Release Notes:** last minute instructions for the end user;
- **Training Materials and Documentation:** materials and documentation provided to help the end user in working with the product created.

Iterative Vs Incremental

When talking about Agile development, most people often use the term Iterative Development instead of Incremental. Despite being somewhat similar, the differences can be easily discovered.

As explained before, the Iterative Development is built on the basis that we will change the software on later analysis, so we are constantly evaluating and improving the quality of our work.

In contrast, the incremental development is meant to be done by continually adding more features to the program until there is the final result.

Another issue that differentiates both methods are the uses of each one.

The iterative is used to improve a "candidate solution" of a problem whereas the Incremental Method is used to gradually develop the functionality of a program.

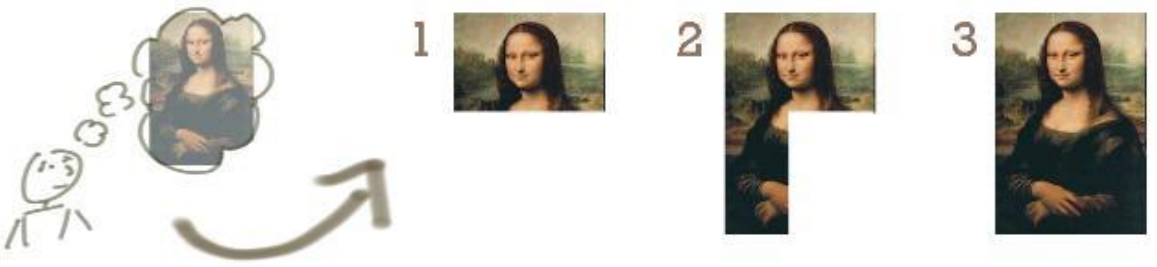

Figure 2: Iterative Method

Figure 3: Incremental Method

Videos/Tutorials:

Iterative and Incremental:

https://www.youtube.com/watch?v=gcpOZi6Hz38&t=19s

Iterative Model (theoretical explanation):

https://www.youtube.com/watch?v=XVvbzycRQ_Y

Agile Software Development - Part 5 - Iterative, incremental and evolutionary:

https://www.youtube.com/watch?v=mUdCDMpKojY

From Incremental & Iterative to Agile - What's the Right Process for Your Team?:

https://www.youtube.com/watch?v=Vlc2r_U30yo&t=384s

Online Documentation/ Source:

[1] Wikimidea Foundation Inc. "Iterative Design". Wikipedia. https://en.wikipedia.org/wiki/Iterative_design (accessed October 11, 2017).

[2] "What is Iterative model- advantages, disadvantages and when to use it?". ISTQB Exam Certification. http://istqbexamcertification.com/what-is-iterative-model-advantages-disadvantages-and-when-to-use-it/ (accessed October 11, 2017).

[3] Farcic, Viktor. "Software Development Models: Iterative and Incremental Development". Technology Conversations. https://technologyconversations.com/2014/01/21/software-development-models-iterative-and-incremental-development/ (accessed October 12, 2017).

[4] Powell-Morse, Andrew. "Iterative Model: What Is It And When Should You Use It?". Air Brake. https://airbrake.io/blog/sdlc/iterative-model (accessed October 12, 2017).

[5] Aguiar, Ademar. "Software Engineering". Moodle UP. https://moodle.up.pt/pluginfile.php/145186/mod_resource/content/0/4%20MIEIC-ESOF-2017-18-Processes.pdf (accessed October 13, 2017).

[6] Cyclosis. "Methodology". Cyclosis. http://cyclosys.com/practices/methodologiesframework (accessed October 13, 2017).

[7] Scott, Alister. "Iterative vs Incremental Software Development". WatirMelon. https://watirmelon.blog/2015/02/02/iterative-vs-incremental-software-development/ (accessed October 14, 2017).

[8] Patton, Jeff. “Don’t Know What I Want, But I Know How to Get It”. Jeff Patton and Associates. http://jpattonassociates.com/dont_know_what_i_want/ (accessed October 14, 2017).

[9] “Guiding Principles of Iterative Development”. Ihris. https://wiki.ihris.org/wiki/Guiding_Principles_of_Iterative_Development (accessed October 15, 2017).

[10] Basili, Vic and Craig Larman. “History of Iterative”. WikiWikiWeb. http://wiki.c2.com/?HistoryOfIterative (accessed October 17, 2017).

[11] “Iterative Software Development Approach”. NCI Wiki. https://wiki.nci.nih.gov/display/CommonProjects/Iterative+Software+Development+Approach#IterativeSoftwareDevelopmentApproach-ArtifactsofIterativeDesignandDevelopment (accessed October 18, 2017).

[12] Royce, Winston W. “Managing the development of large software systems”. Computer Science, University of Maryland. http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf (accessed October 18, 2017).

2. Team Software Process

Team Software Process, also known as TSP**,** is a software development strategy developed in 1996 by Watts Humphrey.

Figure 4: Watts Humphrey

When a group of engineers start working together, they get little guidance in how to proceed and how to divide their roles in a project development. TSP provides a well-defined process framework that is designed to help teams of managers and software engineers organize their projects and produce software products in a way that improves the quality and productivity of the team, helping them to meet the cost and schedule commitments. The main goal of TSP is to show managers and engineers how to build self-directed teams, in order to plan and manage their work, coach and motivate their teams, all while sustaining peak performance.

To do that, TSP divides the team in various **roles** with specific functions:

- **Team leader:** has to manage project outcomes, solve team and customer issues, talk with the management and also guide and motivate all the team members. One of the most important responsibilities of a team leader is to make sure that the team sticks to the plan they have made in the launch phase. The team leader should also check what issues their members have found and who should handle it, tracking every issue with the issue tracking log (ITL);
- **Team coach:** as the process expert, should focus on supporting and developing the team and all its individuals, while being completely independent of project management responsibility;
- **Quality manager:** helps the team to develop their quality plan and to track process and product quality, helping the team to improve each individual's work;

- **Planning manager:** supports and guides the team in planning and tracking their work, resolving all task dependencies between teammates;
- **Design manager:** guides the team in designing the product, helping to reduce defect injection with high-quality design;
- **Support manager:** helps the team obtain the needed tools to use in the project as well as all the administrative support the team needs;
- **Implementation manager:** responsible for guiding the team in developing the product;
- **Test manager:** is responsible for testing the product being produced;
- **Customer Interface manager:** their focus is the customer that attributed the project to the team.

It is also important to notice that the knowledge and skills learned by training in the Personal Software Process (PSP) are required to use the TSP in the best way possible. PSP training includes learning how to gather and use process data, make detailed plans, manage product quality as well as using earned value to find a project.

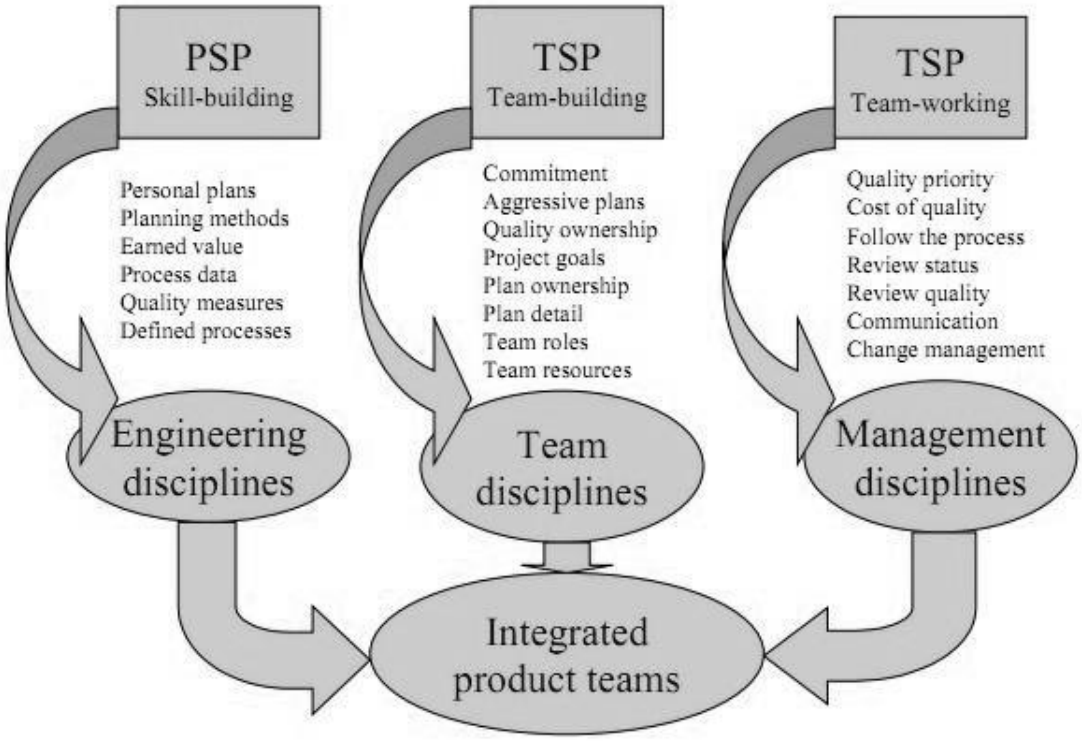

Figure 2: TSP Main Elements

The TSP **software development cycle** can be defined by the following steps:

- **Launch:** a very important part of TSP, consisting in a planning process led by the coach, where the team establishes goals, defines the needed team roles and tasks for each

member, assesses the risks, estimates effort and produces a development strategy. In the end of this step, a management review is held to validate (or not) the work done so far. One key aspect of TSP is that the workload of each team member is taken serious. Everyone as to have a balanced workload so that the team can thrive. Not only that, but sometimes major changes in membership and in the project also happen. This is why sometimes, a single project has to be relaunched several times, at the beginning of each cycle;

- **Requirements:** in this part, the team reflects on what the client wants, producing a development strategy based on these requirements;
- **High Level design:** here, the team designs the project they are responsible for, in the most high-quality and complete way possible;
- **Implementation:** during this step, the developers track the planned and actual effort, schedule, and solve the defects of their work, meeting regularly to revise plans and check the work of everyone;
- **Release test:** testing parts of the project;
- **Post Mortem:** revise planned parameters, assess performances and what was learned to help future process improvement.

With the TSP, teams need to make weekly reports that compare where the team stands against what was planned, as well as complete customer status reports, maintaining management informed about the progress made.

Other important characteristic of TSP is the way teams track progress against what they had planned for each week of work. The method use to do this is called **earned value** and consists of assigning each task a value, based on the time percentage between project time and the time required for that task. For example, a 1000 task hour project with a 45 hours task would have 4.5 value (100*45/1000). When the team completes this task, they earn this 4.5 points. In this way, the team reports that present this earned value can help team members to know where they stand in the project, estimate when they will finish the job and also helps management to understand the job so far and anticipate schedule problems very early, having more time to fix them.

The quality of a project is an essential part of it. The TSP shows teams how to manage quality, by following some topics:

1. **Make a quality plan:** during the launch, teams should make a quality plan, where they estimate how many defects they will inject in each phase of the project, based on the estimated size of the product and historical data on defect injection rates or the TSP quality

guidelines shown in the next figure. With this plan, a team can see if the quality parameters are reasonable, if not, a new quality plan with some adjustments has to be made.

Measure	Goal	Comments
Percent Defect Free (PDF)		
Compile	> 10%	
Unit Test	> 50%	
Integration Test	> 70%	
System Test	> 90%	
Defects/KLOC:		
Total defects injected	75 - 150	If not PSP trained, use 100 to 200.
Compile	< 10	All defects
Unit Test	< 5	All major defects (in source LOC)
Integration Test	< 0.5	All major defects (in source LOC)
System Test	< 0.2	All major defects (in source LOC)
Defect Ratios		
Detailed design review defects /unit test defects	> 2.0	All major defects (in source LOC)
Code review defects/compile defects	> 2.0	All major defects (in source LOC)
Development Time Ratios		
Requirements inspection/requirements time	> 0.25	Elicitation in requirements time
High-level design inspection/high-level design time	> 0.5	Design work only, not studies
Detailed design/coding time	> 1.00	
Detailed design review/detailed design time	> 0.5	
Code review/code time	> 0.5	
Review and Inspection Rates		
Requirements pages/hour	< 2	Single-spaced text pages
High-level design pages/hour	< 5	Formatted design logic
Detailed design text lines/hour	< 100	Pseudocode ~ equal to 3 LOC
Code LOC/hour	< 200	Logical LOC
Defect Injection and Removal Rates		
Requirements defects injected/hour	0.25	Only major defects
Requirements inspection defects removed/hour	0.5	Only major defects
High-level design defects injected/hour	0.25	Only major defects
High-level design inspection defects removed/hour	0.5	Only major defects
Detailed design defects injected/hour	0.75	Only design defects
Detailed design review defects removed/hour	1.5	Only design defects
Detailed design inspection defects removed/hour	0.5	Only design defects
Code defects injected/hour	2.0	All defects
Code review defects removed/hour	4.0	All defects in source LOC
Compile defects injected/hour	0.3	Any defects
Code inspection defects removed/hour	1.0	All defects in source LOC
Unit test defects injected/hour	0.067	Any defects
Phase Yields		
Team requirements inspections	~ 70%	Not counting editorial comments
Design reviews and inspections	~ 70%	Using state analysis, trace tables
Code reviews and inspections	~ 70%	Using personal checklists
Compiling	~ 50%	90+ % of syntax defects
Unit test - at 5 or less defects/KLOC	~ 90%	For high defects/KLOC - 50-75%
Integration and system test - at < 1.0 defects/KLOC	~ 80%	For high defects/KLOC - 30-65%
Before compile	>75%	Assuming sound design methods
Before unit test	> 85%	Assuming logic checks in reviews
Before integration test	> 97.5%	For small products, 1 defect max.
Before system test	> 99%	For small products, 1 defect max.

Figure 3: TSP Quality Guidelines

2. **Identifying Quality Problems:** the best way to find quality problems is to compare the current project with similar ones and see what were the parts that instigate more defects. With that thinking, TSP introduces some quality measures that are:

- **Percent defect Free (PDF):** usually, a PDF plot is built, showing the percentage of the system's components that didn't have defects found in a particular defect removal phase, helping to find a particular harder phase.
- **Defect-removal profile:** while the PDF plot can only be made for an overall system or large component, the defect-removal profile can be used for the whole system, its subsystems or even small components, allowing a progressively examination of the system to find the source of the problem.

- **Quality profile:** this measures the process data for a module against the organization's quality standards, using their existing historical data to produce standards or the TSP quality guidelines (when that historical data is not enough). A quality profile consists of the following dimensions: design and code review time, compile and unit tests defects and design and code production time.
- **Process quality index (PQI):** a single numeric quality figure that results by taking the product of the dimensions of the quality profile, evaluating how a module is defect free.

3. **Prevent Quality problems:** Once a TSP team has identified the components that most likely have quality problems, they should do as follows:

- Monitor the module during test to see if problems are found and then determine the remedial action;
- Re-inspect the module before integration or system test;
- Have an engineer rework the module to fix suspected problems;
- Redevelop the module.

In this way, TSP processes are designed to prevent defects before they even occur, having always an eye for the development of the current project and the one's after that, by building defect reviews during the project.

TSP is being introduced into both industrial and academic environments. Some examples of projects where TSP was successfully used can be seen here:

- Teradyne found that, prior to the TSP, defect levels in integration test, system test, field testing, and customer use averaged about 20 defects per KLOC (1,000 lines of code, or LOC). The first TSP project reduced these levels to 1 defect per KLOC. Since it cost an average of 12 engineering hours to find and fix each defect, Teradyne saved 228 engineering hours for every 1,000 LOC of program developed. Since the typical cost to code and unit test 1,000 LOC is about 50 engineering hours, the savings in defect repair costs were about 4.5 times the cost of producing the programs in the first place.
- The first TSP project at Hill Air Force Base, near Salt Lake City, found that team productivity improved 123% and test time was reduced from an organizational average of 22% to 2.7% of the project schedule.
- Nedbank, one of the largest banks in South Africa, in collaboration with the SEI and the Johannesburg Centre for Software Engineering at the University of Witwatersrand, used TSP to improve software performance at the individual and team levels. SEI worked with Nedbank to address the challenges of expanding and scaling the use of TSP to an

organizational level. SEI researchers also helped Nedbank explore challenges common to many organizations seeking both to improve performance and to become more agile.

- Beckman Coulter, a company that manufactures biomedical testing instrument systems, introduced TSP in a push for "differentiated quality" in the marketplace. While Beckman Coulter used TSP for new development, they also tried something new: applying TSP late in development to a project already in system test. This highly successful effort resulted in a factor of 10 improvement in system reliability, satisfied customers noting no major issues after release, and a highly predictable schedule (only 3.4% variance). Beckman Coulter's TSP projects have demonstrated at least 5 times fewer defects after release.

Reading all of this, we can conclude that TSP has many proven **benefits**, like improved size, effort and schedule estimation, improved productivity, better defect detection as well as higher quality products. However, TSP is a very **strictly** approach to software development, having the need to do a lot of plans that **have** to be followed, having weekly meetings, think a lot before actually work and a it puts a lot of pressure from management side to team members.

Without a doubt, TSP is a software development strategy that differentiates from others because of the main role a **team** has in the software development. More than focusing on the individual, TSP forms a **strong team** with well-defined **roles**, in order to produce the **best software possible** for the clients it attends.

Videos:

Team Software Process (explanation):

https://www.youtube.com/watch?v=pNaUkHdfEd8

[1] Wikipedia Foundation Inc. “Team Software Process”. Wikipedia. https://en.wikipedia.org/Team_software_process (accessed October 11, 2017).

[2] Humphrey, Watts “The Team Software Process^{SM} (TSP^{SM})” Software Engineering Institute. https://resources.sei.cmu.edu/asset_files/TechnicalReport/2000_005_001_13754.pdf (accessed October 15, 2017).

[3] “Team Software Process^{SM}” Flylib. http://flylib.com/books/en/4.421.1.24/1/ (accessed October 15, 2017).

[4] “Case Studies”. Software Engineering Institute. https://www.sei.cmu.edu/tsp/casestudies/ (accessed October 15, 2017).

[5] Rugaber, Spencer “Team Software Process (TSP)” http://www.star.cc.gatech.edu/documents/SpencerRugabear/tsp.pdf (accessed October 16, 2017).