Fast development Extreme Programming (XP)

Yohan Carlos Camacho Santana, Fidelio Castillo Romero

[1] Tecnológico Nacional de México / Instituto Tecnológico de Villahermosa, Tabasco

Abstract

Developing software has always been a great challenge. Proof of this are the numerous methodological proposals that transgress in the various areas of the process. On the one hand, we find more traditional proposals focused on controlling the process, closely establishing the activities involved, the artefacts and tools, as well as the notations to be used. These approaches have proven effective and useful in countless projects, but have also exhibited problems in others. An accepted refinement is to include more activities, artefacts and constraints in the development processes, based on the weaknesses observed. However, the end result would be a much more problematic development process and may restrict the team's own ability to deliver the project. Another approach is to focus on other dimensions, such as human resources or the software product. Agile methodologies are based on this ideology, which place greater value on the individual, customer support and incremental software development, using very short iterations. This approach is proving its worth in projects with changing requirements and when there is a tendency to reduce development times, without neglecting the final quality of the product. Agile methodologies are setting trends in software production, and at the same time creating a deep debate between their followers and those who, due to certainty or distrust, do not perceive them as possible alternatives to traditional methodologies. This paper summarizes the emergence of agile methodologies, their values, principles and some comparisons with traditional methodologies. At the same time, eXtreme Programming (XP), the most popular agile methodology today, is described almost in cellular form.

Resumen

Desarrollar software siempre ha sido un gran desafío. Prueba de ello son las numerosas propuestas metodológicas que transgreden en los distintos ámbitos del proceso. Por un lado, encontramos propuestas más tradicionales enfocadas a controlar el proceso, estableciendo de cerca las actividades involucradas, los artefactos y herramientas, así como las notaciones a utilizar. Estos enfoques han demostrado ser eficaces y útiles en innumerables proyectos, pero también han presentado problemas en otros. Un refinamiento aceptado es incluir más actividades, artefactos y limitaciones en los procesos de desarrollo, en base a las debilidades observadas. Sin embargo, el resultado final sería un proceso de desarrollo mucho más problemático y podría restringir la propia capacidad del equipo para ejecutar el proyecto. Otro enfoque es centrarse en otras dimensiones, como los recursos humanos o el producto de software. Las metodologías ágiles se basan en esta ideología, que dan mayor valor a la persona, la atención al cliente y el desarrollo de software incremental, utilizando iteraciones muy cortas. Este enfoque está demostrando su valor en proyectos con requisitos cambiantes y cuando existe una tendencia a reducir los tiempos de desarrollo, sin descuidar la calidad final del producto. Las metodologías ágiles están marcando tendencia en la producción de software, y al mismo tiempo creando un profundo debate entre sus seguidores y quienes por certeza o desconfianza, no las perciben como posibles alternativas a las metodologías tradicionales. Este artículo resume el surgimiento de metodologías ágiles, sus valores, principios y algunas comparaciones con las metodologías tradicionales. Al mismo tiempo, eXtreme Programming (XP), la metodología ágil más popular en la actualidad, se describe casi en forma celular.

Keywords: Programming methodology, Agile methodology, Software processes, Extreme Programming.
Palabras claves: Metodología de la programación, Metodología Ágil, Procesos de software, Programación Extrema.

1. INTRODUCTION

In the last decades modelling notations and consequently tools proved to be the safeguards for successful software development, however, expectations were not met. This is largely due to the fact that the development methodology has been neglected. Excellent notation and tools are of little use if the guidelines for their implementation are not met. Thus, this decade has awakened with a gradual increase of interest in development methodologies. A few years ago, the development process had a strong emphasis on process control, rigidly defining roles, activities and artefacts, containing scrupulous modelling and documentation. This "traditional" representation for undertaking software development has proven to be effective in large projects (in terms of time and resources), where a high degree of process coupling is required. This approach does not fit in many of the usual projects where there is instability in the system environment due to the volatility of the changes it registers, and there is a demand to drastically reduce development times, without neglecting the quality that is needed. In this scenario, agile methodologies germinate as a possible answer to these problems. Because they are essentially aimed at small projects, agile methodologies establish a tailor-made solution, without renouncing the good practices that guarantee the quality of the product. In recent years, the curiosity shown by most software engineers and teachers about agile methodologies has led to a strong projection towards industry. On the one hand, for many development teams, the use of traditional methodologies is cumbersome and far removed from their way of working, considering the conflicts of their introduction and the investment involved in training. On the other hand, the typologies of projects for which agile methodologies have been essentially premeditated. They fit a wide range of industrial software projects; those with a small development team, generally short development times, unstable requirements, and/or based on new technologies.

2. DEVELOPMENT

Agile methodologies

In February 2001 in the city of Utah-USA, the term "agile" applied to software development was born. A group of 17 experts from the software industry, including some of the creators of software methodologies, participated in this meeting. The objective was to outline the values that should make it easier for teams to develop software, responding to the changing requirements that may emerge during the life of the project. It was intended to offer a disjunctive to traditional software development processes, distinguished by being rigorous and driven almost entirely by nascent documentation in each of the activities carried out. Several of these agile methodologies had already proven their worth by being successfully applied in real projects, but there was little recognition and dissemination of these methodologies. Following this meeting, The Agile Alliance was created as a non-profit organization, dedicated to promoting the concepts consistent with agile software development and helping organizations to be able to assimilate these concepts. The starting point was the Agile Manifesto (Agile Alliance 2015), a document that summarizes the "agile" philosophy. It includes a series of rules that must be followed by any methodology of this type. As time went by, several methodologies of this type were born, with a group of them standing out.

Review of methodologies

Before summarizing some agile methodologies, let us illustrate the primary differences with respect to traditional ("non-agile") methodologies. The following table outlines these differences schematically, not only taking into account the process itself, but also the team context and structure more favorable to each of these philosophies.

Agile Methodologies	Traditional Methodologies
Less controlled process, with a shortage of principles	Much more controlled process, with numerous policies/regulations
No traditional contract or at least it is quite flexible.	There is a fixed contract
The client is part of the development team.	The customer interacts with the development team through meetings
Small groups (<10 members) and working on the same site	Large and possibly distributed groups
Few artefacts and roles	Bulky artefacts and roles
Less emphasis on software architecture	The software architecture is essential and is expressed through models
Based on heuristics from code production practices	Based on norms derived from standards followed by the development environment.
Specially prepared for changes during the project	Some resistance to change
Internally imposed (by the development team)	Externally imposed

Table 1. Differences between methodologies in general terms.

Each methodology has its own characteristics and emphasizes some of the more specific aspects. Some agile methodologies are summarized below, postponing the more exhaustive analysis of XP to the next section.

Crystal Methodologies (Cockburn 2021). These are a set of methodologies distinguished by their focus on the people who make up the team (the success of the project depends on them) and the minimization of the number of artefacts produced. Alistair Cockburn was the forerunner and creator of this philosophy. The development team is a key factor, so efforts must be invested in improving their skills and abilities, as well as having defined teamwork policies. Policies will depend on the size of the team, classified by colour, e.g., Crystal Orange (25 to 50 members).

Adaptive Software Development (ASD) (Highsmith 2013). Its creator is Jim Highsmith. It consists mainly of the following characteristics: software component-oriented rather than task-oriented, highly change-tolerant and iterative. It exhibits a life cycle with three essential phases: learning, collaboration and speculation. In the speculative phase, the project is initiated and the software features are planned; in the collaborative phase, the features are developed and finally in the last phase, the quality aspects are developed and the project is delivered. The review of the components is useful to learn from mistakes and restart the development cycle.

SCRUM (Schwaberand Beedle 2021). It defines a framework for project management, which has been used successfully for the last 10 years. It is especially configured for projects with high instability in requirements. One of its main characteristics is the realization of software development through iterations, which are called sprints, with a life time of 30 days. The result of each sprint is shown to the client. Another important feature is the meetings throughout the project. The daily 15-minute meeting of the development team for coordination is one of the most important.

Dynamic Systems Development Method (DSDM) (Stapleton & Constable 1997). It was born in 1994 with the aim of creating a unified methodology. Among its main characteristics are: the development team and the user work together and it is an iterative and incremental process. It proposes five phases: functional modelling, feasibility study, design and construction, business study and implementation. In addition, there is feedback to all phases.

Lean Development (LD) (Poppendieck and Poppendieck 2003). Defined by Bob Charrette's from experiences gained in the development of projects in the Japanese automotive industry and used in countless telecommunications projects in Europe. In ML, change is seen as a risk, but if managed appropriately it has the potential to become an opportunity that can improve the productivity of the client.

We will carry out a comparison of the different agile approaches based on three parameters: view of the system as something changing and more specific characteristics of the methodology itself, such as: technical excellence, results, simplicity, etc. The Capability Madurity Model (CMM) is also added as a non-agile reference.

Features	Methodologies						
	ASD	**CRYSTAL**	**DSDM**	**LD**	**SCRUM**	**XP**	**MWC**
Changing System	5				5	5	1
Collaboration	5	5			5	5	
Characteristics-Methodology							
Results	5	5			5	5	
Simplicity					5	5	1
Adaptability	5	5					
Technical Excellence							
Collaborative practices	5	5				5	
Media CM	4.4	4.4	3.6	3.6	4.2	4.4	2.2
Average Total	4.8	4.5	3.6	3.9	4.7	4.8	1.7

Table 2. Agility Ranking (Highsmith 2002).

From the analysis outlined above, it can be summarized that all agile methodologies show a marked difference in agility index with respect to CMM, with ASD, Scrum and XP standing out as the most agile.

EXTREME PROGRAMMING, (XP)

XP (Beck and Andres 2005) is an agile methodology focused on solidifying interpersonal relationships, on the basis that these are the key to successful software development, promoting teamwork and fostering a favorable working environment. XP is based on the continuous feedback that must exist between the client and the development team, simplicity in the solutions made and boldness to face changes. XP is defined as suitable for projects with indeterminate requirements, with a high probability of changes occurring and where high technical risk coexists. Kent Beck is the creator of this methodology, his great vision of how to face the changing challenges of an unstable project makes us currently enjoy a very effective way of planning, developing and implementing a project with this characteristic. The essential characteristics that make up the XP methodology present a high level of importance to understand how to apply it, so we will organize them in the following three sections: user stories, roles, process and practices.

User Stories

User stories are the technique used in XP to specify software requirements. They are paper cards on which the customer describes in brief the particularities that the system must have, whether functional or non-functional requirements. The treatment of user stories is very dynamic and flexible, at any time a user story can be replaced by more specific ones, a new one can be added or an existing one can be modified. Each user story is sufficiently understandable and delimited so that programmers can complete them in a few weeks (Jeffries, Anderson & Hendrickson 2001). In terms of what information, a story should contain, there are a variety of suggested outlines, but no consensus has been reached. In some cases, it is suggested to provide a name and a description (Wake 2002) or to dispense with the former, perhaps adding an estimate of effort in days

ISSN: 2007-4786
Volumen 13 – Número 3
Julio – Septiembre 2021

(Newkirk & Martin 2001). Beck in his book (Beck & Andres 2005) presents an example of a customer story and task card in which the following statements are illustrated: date, type of activity (new, correction, improvement), functional test, story number, technical and customer priority, reference to another previous story, risk, technical estimate, description, notes and a follow-up list with date, status of things to be completed and comments. Another unknown that arises may be the level of granularity that a user story should cover. Jeffries in (Jeffries, Anderson, y Hendrickson 2001) says that it is linked to the complexity of the system, there should be at least one story for each major feature, and his proposal is to do one or two stories per programmer per month. If you have fewer, it is probably wise to split the stories, if you have more it is best to reduce the detail and group them together. User stories are broken down into programming tasks and assigned to programmers to be implemented during an iteration.

XP Roles

Despite the variations and extensions of XP roles that are reflected in various sources, in the following section we will unpack the roles based on the original proposal of the creator of the methodology.

Programmer: The programmer writes the unit tests and produces the system code. There must be close communication between programmers and other team members.

Client: The client writes the user stories and tests to validate their implementation. Likewise, it assigns priority to the user stories and decides which are implemented in each iteration, focusing on providing greater value to the business. The client is only one within the project, and may be the representative of a company or entity.

Tester: The tester assists the customer in writing functional tests. He/she executes the tests, disseminates the results to the team and is responsible for the test support tools.

Tracker: The tracker provides feedback to the team in the XP process. His commitment is to identify the degree of accuracy between the estimates made and the actual time spent, communicating the results to improve future estimates. In addition, he/she must track the progress of each iteration. He determines when changes are needed to achieve the objectives of each iteration.

Coach: Has overall responsibility for the process, attends and manages the entire project. It is necessary that he/she has a thorough knowledge of the XP process to provide guidance to the team members so that XP practices are applied and the process is followed correctly.

Consultant: This is an external member of the team with expertise in some area of knowledge with the aim of clarifying specific issues for the project.

Big boss: The link between clients and programmers, helping the team to work effectively by putting the right conditions in place. Coordination is the fundamental task of this role.

XP Process

Victory in an XP project is achieved when the customer chooses the business value to implement based on the team's expertise in measuring the functionality it manages to deliver over time. The following outlines the XP development cycle: (Jeffries et al. 2001)

1 The customer defines the business value to be implemented.
2 The programmer estimates the effort required for implementation.
3 The client chooses what to create, according to his priorities and time conditions.
4 The programmer builds that business value.
5 Back to step 1.

The developer should not be pressured to perform work in excess of the estimate, as there is a risk of reducing the quality of the final product or missing the delivery deadline. The customer has the obligation to maneuver the product delivery scope, ensuring for himself that the system achieves the highest business value per iteration. The ideal XP lifecycle consists of six phases (Beck andAndres2005): Exploration, Release, Iterations, Production, Maintenance and Project Dead.

Phase I: Exploration

In this phase, customers present user stories that are of interest for the first delivery of the product. In turn, the development team familiarizes itself with the tools, technologies and practices that will be used in the project. A prototype is built in order to test the technology and explore the possibilities of the system architecture. The duration of the exploration phase ranges from a few weeks to a few months, depending on how familiar the programmers are with the technology.

Phase II: Delivery Planning

The client in this phase establishes the priority of each user story, then the programmers make an estimate of the effort involved for each of them. A schedule is determined together with the client to agree on the delivery. Delivery should be achieved in less than three months. Estimates of the effort associated with the execution of the stories are made by the programmers using the point as a measure. One point is equivalent to an ideal week of programming. Stories are generally worth 1 to 3 points. The duration of this phase is a few days. A record is kept of the "speed" of development, set in points per iteration. The speed of the project is used to constitute how many stories they are able to implement before a certain date.

Phase III: Iterations

This phase includes several iterations on the system before it is delivered. The Delivery Plan is composed of iterations that do not exceed three weeks. In the first iteration, an attempt can be made to establish a system architecture that can be used for the remainder of the project. At the end of the last iteration the system should be ready to go into production. Elements needed during the development of the Iteration Plan are: unaddressed user stories, project velocity, unsuccessful acceptance tests and unfinished tasks from the previous iteration. All iteration work is indicated in scheduling tasks, each of which is assigned to a programmer to be responsible for, but is solved in pairs. Wake in (Wake 2002) provides a few guidelines for committing delivery planning and each of the iterations.

Phase IV: Production

This phase requires additional testing and performance reviews before the system is relocated to the customer environment. Simultaneously, decisions must be made about adding new features to the current version due to changes during this phase. It is possible that the time for each iteration will decrease from three weeks to one week. Ideas that have been proposed and suggestions are documented for further implementation.

Phase V: Maintenance

While the first version is in production, the XP project must support the running system in conjunction with the development of new iterations. In order to carry out this process, customer support tasks are required. In this way, the speed of development can plummet after the system is put into production. The maintenance phase may require new staff and changes in its structure.

Phase VI: Death of the Project

This is when the customer does not submit any more stories to be included in the system. This requires that the customer's needs are met in other aspects such as system performance and reliability. The final documentation of the system is created and no further changes are made to the architecture. Project death also occurs when the system does not generate the benefits expected by the customer or when the budget is tight.

3.- CONCLUSIONS

There is no universal methodology for tackling any development project. Any methodology must be adapted to the context of the project. It has been proven that traditional methodologies have tried to address as many project context situations as possible, requiring a considerable effort to be adapted, especially in small projects and with very changing requirements. Agile methodologies, on the other hand, offer an almost tailor-made solution for a large number of projects where these characteristics are present. One of the most praiseworthy qualities of an agile methodology is its simplicity, both in its application and in the learning effort. This has led to an increased interest in agile methodologies. However, some drawbacks and restrictions to their application should not be forgotten, such as: they are aimed at small or medium-sized teams, the physical environment must be an environment that enables communication between all team members during the life of the project, any resistance from the client or the development team to the practices and principles can lead the process to failure, among others. There is a lack of consensus on the theoretical and practical aspects of using agile methodologies, as well as a lack of further consolidation of application results. Research activity is oriented towards lines such as: process metrics and evaluation, specific tools to support agile practices, human and teamwork aspects. Although there are currently many books that explain the principles and rules of each of the existing methodologies and there is a wealth of information on the Internet, XP is the methodology that stands out for having the largest amount of information available, being by far the most popular among the general public.

REFERENCES

[1] Agile Alliance. 2015. "About Agile Alliance. Agile Alliance |. Retrieved 14 April 2021 (https://www.agilealliance.org/the-alliance/).
[2] Beck, Kent and Cynthia Andres. 2005. Extreme Programming Explained Embrace Change. 2nd ed. Boston: Addison-Wesley.
[3] Cockburn, Alistair. 2021. Agile Software Development. Vol. 1. Harlow: Addison-Wesley.
[4] Highsmith, James A. 2002. Agile Software Development Ecosystems. Boston: Addison-Wesley.
[5] Highsmith, James A. 2013. Adaptive Software Development a Collaborative Approach to Managing Complex Systems. Addison-Wesley Professional.
[6] Jeffries, Ron, Ann Anderson and Chet Hendrickson. 2001. Extreme Programming Installed. Boston: Addison-Wesley.
[7] Newkirk, James W. and Robert C. Martin. 2001. Extreme Programming in Practice. Addison-Wesley Professional.

[8] Poppendieck, Mary and Tom Poppendieck. 2003. Lean Software Development an Agile Toolkit for Software Development Managers. 1st ed. Addison-Wesley Professional.
[9] Schwaber, Ken, and Mike Beedle. 2021. Agile Software Development with SCRUM. Prentice Hall.
[10] Stapleton, Jennifer and Peter Constable. 1997. DSDM, Dynamic Systems Development Method. Harlow: Addison-Wesley.
[11] Wake, William C. 2002. Extreme Programming Explored. Boston: Addison-Wesley.

Author Email: *yccamacho87@aol.com*