Scrum-based Methodology for Distributed Software Development

Eva del Nuevo, Mario Piattini
Alarcos Research Group
University of Castilla La Mancha
Ciudad Real, Spain
{Eva.Nuevo, Mario.Piattini}@uclm.es

Francisco J. Pino
IDIS Research Group
University of Cauca
Popayán, Colombia
fjpino@unicauca.edu.co

Abstract—**There are many companies doing software development in a distributed way nowadays, due to the great benefits that it provides; however, this type of development also leads to multiple complications such as deteriorated communications. Among the most widespread methodologies for software development is the Rational Unified Process (RUP). However, in recent decades a series of methodologies have emerged, called "agile methodologies", which aim to develop software quickly, focusing on people and frequent delivery of software. Of the existing agile methodologies, Scrum is one of the most widely-applied, due to its ability to complement other methods and processes. For this reason, the strategies proposed by Scrum may be suitable for the distributed management and deployment of the phases and disciplines of RUP. In this paper we present a methodology based on the integration of RUP and Scrum, called *scRumUP*, which has been so designed as to be appropriate in a distributed environment.**

Distributed software development; Agile software development; Agile methods; Agile methodologies; Scrum

I. Introduction

Companies located in different places are currently looking for cost savings and are seeking to be competitive in the development of software products. However, to ensure the quality of the products developed, these companies need to have a methodology which provides elements to face the challenges involved in Global Software Development (GSD). Approaches widely used and accepted by the software industry, such as Scrum and the Rational Unified Process (RUP), could be used to propose a methodology for supporting and guiding GSD. RUP is a generic framework for iterative and incremental software development that can be modified to adapt to different development needs; and Scrum is an iterative and incremental generic framework for project management and agile software development, which has the ability to complement other methods and processes. In this regard, a methodology that integrates elements from these approaches can offer a complete and robust proposal for addressing GSD. It can, moreover, be strengthened by the different elements from these two approaches.

Taking into account the aspects described above, this paper proposes *scRumUP*, a methodology for global software development that incorporates and integrates into the process management described by Scrum the concepts of software development proposed by the Rational Unified Process. This methodology has been designed to be applied in a distributed environment, and its purpose is to offer guidelines for the management and development of software in a distributed environment, using the advantages provided by the integration of agile methods such as Scrum and traditional methodologies such as the Rational Unified Process. Therefore, *scRumUP* is aimed at companies doing GSD that want to improve the management of their projects and reduce the complications resulting from the distribution by the application of agile methods. Although it is not necessary for the company to be applying agile methods, in that case the transition would be smoother. In addition, *scRumUP* has been adapted to that of a Spanish software development company, which is currently using it in a pilot project.

The document is structured as follows. Section II presents the related work on agile methods in distributed environments. Section III describes the methodology that we have developed and proposed for guiding GSD. Section IV presents the adaptation of this methodology to that of the software development company. Finally, the conclusions and future work are outlined in Section V.

II. Related Work

Although we know there are articles dealing with the integration of heavy and light methods, this section will only focus on analyzing various articles that address the topic of agile methods in GSD.

There are several articles dealing with agile methods in distributed environments. In [1], some best practices for distributed agile teams are presented, such as organizing the team structure around the architecture, rotating the times of day when the coordination meetings are held, and having ambassadors and boundary spanners. The author of [2] divides agile practices into three categories: feasible, infeasible and modified. He concludes that agile practices need to be modified for outsourced software projects and that research into the subject could lead to a hybrid approach that harmonizes agility and discipline. In [3] a case study that uses their own methodology based on Scrum and Feature-Driven Development (FDD) is presented. However, the methodology is not explicitly defined, placing more emphasis on the coordination techniques used in the sprint (shared mailing list, status emails, daily meetings and CVS repository), and in the lessons learned from the application of the methodology to a real case, than on the methodology itself. The study described in [4] presents several guidelines for achieving agility in global software development, such as acquiring up-to-date knowledge on new technologies, clearly defining roles and responsibilities, or standardizing the IT platform. The authors in [5] suggest a framework for software globalization project management using a distributed agile approach by means of an industrial case study that shows successfully integrated practices from agile software development and distributed software development. In [6] two distributed team structures (partial offshoring and complete offshoring) are proposed. In the first one, the onshore team is the design team and sends the requirements to the offshore team, which responds with a fully-tested working code. In the second one, the only person left onshore is the customer, who sends the features to the offshore team, which responds with a fully-tested working code. The study shown in [7] recommends best practices for globally distributed teams, such as hourly automated builds from one central repository, seamless integration of XP practices, like pair programming with Scrum, or the writing of the three scrum questions before the meeting.

The main difference between the papers mentioned above and our paper is that none of these papers explicitly proposes a process that describes how to apply agile methods to distributed software development. Most of these papers recommend practices and techniques that have been successfully applied in a real case, but without specifying in detail what activities and tasks have been carried out. From this analysis of the literature, which includes the papers mentioned above as well as other papers related to the subject, we found the need to define a methodology that joins the benefits provided by agile methods to a software development process, such as RUP. Our contribution is therefore a hybrid proposal that performs the project management following Scrum and the software development following RUP. It also uses the agile practices Test-Driven Development (TDD) and Continuous Integration (CI) during development.

III. Proposed Methodology

We describe *scRumUP* here, by presenting first of all the conceptual support on which it was developed, then the distributed structure, and finally the process proposed by the methodology.

A. Conceptual Support

This methodology is based on the Scrum Guide [8] and on previous research in articles and books on the subject, as well as on the Rational Unified Process [9]. We have taken special care to follow many of the recommendations provided in [10] on best practices for scaling agile development, as well as other findings [11] and the systematic review that we have conducted on the topic.

One of the most important elements of distributed development may be how to manage distributed teams, since this affects the productivity of those teams and thus the product development itself. Interaction and coordination are also factors to be taken into account if we want successful distributed software development.

1) Structure of the teams and distributed sites

Among the methods used to manage distributed teams, the ideal one is Distributed Scrum of Scrums according to [12]. Distributed Scrum of Scrum teams are stable, long-term teams made up of workers (7 ± 2 persons) with multiple skills who are able to do all the work to complete a feature of the Product Backlog but without members who are geographically dispersed. When a team has all the skills needed to complete a feature of the Product Backlog it is called a feature team. A customer-centric feature of the Product Backlog is a unit of functionality from the customer's point of view. Organizing the structure around the job function should be avoided (e.g. analysts in USA, programmers in India), as serious communication problems between the various teams may appear and the integration of the system can be problematic; instead, the team structure should be organized around the architecture (i.e., forming feature teams).

However, if it is not possible to form this type of teams, due to the structure of the organization or organizations involved, the Fully Distributed Scrum method (or dispersed teams) can be chosen, in which teams have the same characteristics as those seen in Distributed Scrum of Scrums, but with the difference that in this case, members of a team may be distributed geographically (for more information on team building, see section III.B.1.Formation of distributed teams).

2) Interaction and coordination

The best way of interaction is face-to-face conversation (one of the key agile principles), and to carry this out in a distributed environment, the use of videoconferencing tools is advisable, as eye contact can collect additional information that is very important when interpreting the communication. Videoconference meetings can be started informally to foster social relationships between the teams. Social relationships among team members may also be reinforced through visits to other teams or by sending ambassadors. Another highly recommended practice to create links that support communication among dispersed team members is to start the distributed agile development in a co-located way (all team members together) at the beginning of the project.

Coordination between distributed teams is achieved through frequent meetings. Wherever problems caused by time differences arise, other methods for conducting these meetings should be tried (see section III.B.2.Distributed Meetings).

3) Practices

The following practices are important for achieving agility and coordination in the implementation of *scRumUP*.

- Test-Driven Development (TDD): For each unit of functionality of the code, a test is written first. The test is run to see if it fails, which would show that the property we are trying to implement is not implemented yet. Then only the code necessary to pass the test is produced and all the tests are run (the one that we have just created, along with other existing tests). If the tests are successful, the next step is to refactor the code.

- Continuous Integration (CI): According to [13], "Continuous Integration is a software development practice where members of a team integrate their work frequently; usually each person integrates at least daily, leading to multiple integrations per day. Each integration is verified by an automated construction that includes tests to detect integration errors as quickly as possible".

B. *Distributed Structure*

Dealing with distributed teams means having to face a number of challenges that can negatively impact product development. If the time zone difference between teams is very large, there may be no overlapping work hours between some teams and it may not be possible to conduct frequent meetings for the proper coordination of the work. This lack of meetings must be supplemented with more documentation (in a form that all members can see, e.g. a Wiki) so that all teams can successfully follow the project.

If there is overlap of working hours and it is possible to conduct the meetings, we recommend the use of videoconferencing, as face to face communication is one of the key agile principles. Whenever videoconferencing is not possible, due to technical or bandwidth problems, the telephone can be used as an alternative.

1) *Formation of distributed teams*

According to [14] and [10], there are several ways to make the transition toward feature teams: big-bang reorganization, gradual expansion of team responsibility and gradual introduction of feature teams applied only to the most important features of the Product Backlog.

- Big-bang reorganization: This reorganizes the teams, by grouping various specialists on the same team so that the new teams have knowledge of most of the system. This strategy can work well, but it involves some risk, since the change from component teams (consisting of specialists in an area) to feature teams is big.
- Gradual expansion of team responsibility: This approach makes a gradual transition through small steps, to expand the team's responsibility from single component teams to multi-component teams, finally reaching feature teams. This strategy does not work as well as big-bang organization, because it has the disadvantages both of the component teams and of the feature teams.
- Gradual introduction of feature teams applied only to the most important features: This strategy involves the gradual introduction of feature teams applied only to the most important features of the Product Backlog. However, this approach has disadvantages too, because the feature team may need to make changes to the code of the component teams, meaning that component teams lose control over some of the features. Furthermore, the transition to feature teams with this strategy is slow.

These three forms of making the transition can be applied whether the team is dispersed (with members in different geographic locations) or not. If there are component teams in different geographical locations in the organization, it would be advisable for the teams formed after the transition to feature teams to be co-located, that is, with all the members in the same geographical location. In those cases where dispersed teams are the only choice, members of the same team should be co-located for the duration of the first sprint, at least.

2) *Distributed meetings*

If there are overlapping working hours among the teams, distributed meetings may be conducted during these hours, using tools such as videoconferencing or the telephone. However, if there are no overlapping working hours, other methods will have to be applied to conduct the meetings. According to [11], there are several methods for dealing with distributed meetings:

- Meetings through documentation: This method schedules the meeting at a time when most members can attend the meeting. The meeting will be documented in a Wiki, to inform members that can not be present. This is the least effective way of managing distributed meetings.
- Liaison approach: With this approach, two meetings are held at different times and there is a link person who attends both of them. The link shares information between the members of the two areas.
- Alternating meeting times: This method alternates the schedule of the meetings, so that the meeting sometimes takes place during normal working hours of one team, and other times during normal working hours of the other team/s.
- Share the pain: This approach selects the schedule of the meeting that best suits the team as a whole. In this way, each team has flexible working hours. To choose the time of the meeting, each team member is asked the following questions: What time is best for you to meet? What times are you willing to meet? What times are off-limits? The resulting schedule is the one that works best for the team.
- Feel the pain: In this case, the meetings are always held outside the working hours of one of the teams, i.e. there is one team that "feels the pain" of having to work outside working hours.

C. *Proposed Process*

The development methodology for distributed environments that we propose describes a process that aims to guide and organize the work involved in the development of software products in distributed environments. This process consists of two different parts (see Fig. 1).

The first phase of the process is the Release Planning phase. In this phase the initial planning of the release is

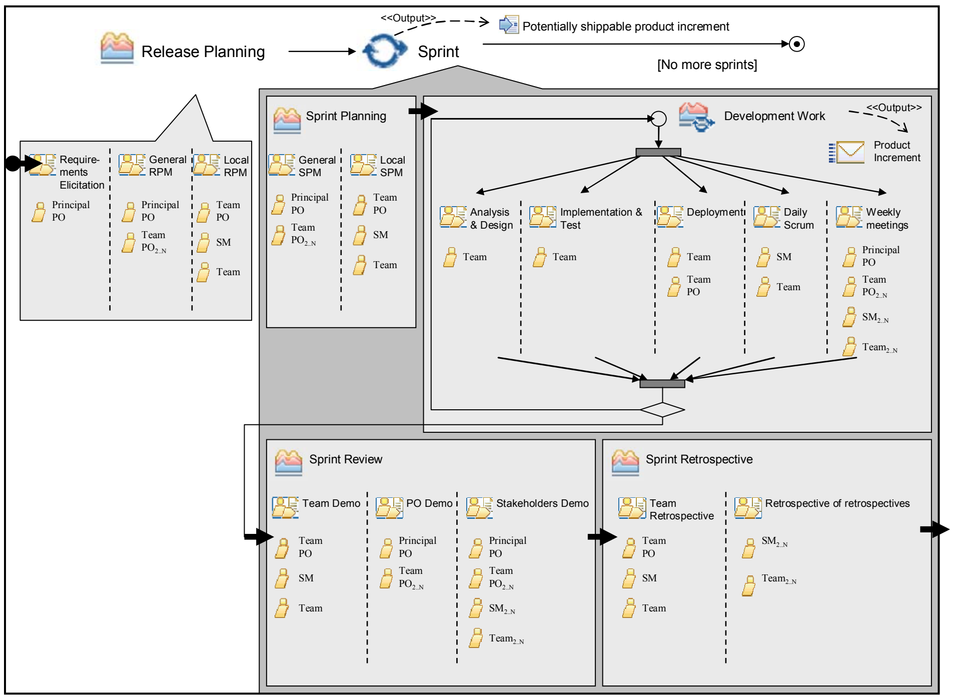


Figure 1. Phases, activities and roles of the process

performed; however, this plan may change over time through adjustments made in other subsequent planning activities. This phase consists of three activities: Requirements elicitation, General Release Planning Meeting and Local Release Planning Meeting. The activity Requirements elicitation involves the Principal Product Owner and collects the stakeholders' needs as requirements, by means of some requirements capture technique (e.g. user stories). In the General Release Planning Meeting activity the Product Owners define the vision of the product, and create and prioritize the Product Backlog, i.e. the artifact that will store the product features. The Product Owners should decide what Product Backlog features have higher priority, based on information provided by stakeholders and their personal criteria. Although there are several ways to manage the Product Backlog [11], in *scRumUP* the use of a single backlog with sections for multiple teams has been considered, where all the members share the same Product Backlog and the Principal Product Owner divides that Product Backlog so that each section is assigned to a team. The last activity of this phase is the Local Release Planning Meeting, in which the Team Product Owner informs the team of the Product Backlog section that has been assigned to them, and then the team estimates the features of that section.

The next element of the process is the sprint, which is an iteration that results in a potentially shippable product increment. It consists of four phases: Sprint Planning, Development Work, Sprint Review and Sprint Retrospective.

The Sprint Planning phase aims to identify which part of the Product Backlog will become working software in this sprint. It is a phase which consists of two activities. The first one is the General Sprint Planning Meeting, where the Principal Product Owner and the Team Product Owners meet to decide what work will be carried out in the sprint. Finally, the Product Owners set the goal of the sprint and identify the features that each team must complete during that phase, so that the dependencies between the features of the teams are minimal. In the Local Sprint Planning Meeting activity the Team Product Owner meets his own team (including the ScrumMaster). They first identify the tasks, i.e. detailed work pieces needed to turn the Product Backlog into working software that must be broken down in order to be made in less than a day. They estimate the number of hours needed to carry out each task and assign these tasks to individual team members according to the particular skills of each one. Then the Sprint Backlog and the Sprint Burndown are created from the tasks identified.

The Development Work phase (see Fig. 2) consists of several activities carried out in an iterative and incremental way, as in the Rational Unified Process. The end of this phase results in a potentially shippable product increment that consists of an executable code, along with the models and manuals related to this code.

The first activity of this phase is Analysis and Design. During analysis the requirements are analyzed in detail, to address issues not treated yet, thereby achieving a better understanding of these requirements. Unlike the business model, which uses the customer's language, the model created during the analysis uses the language of developers, so it is possible to reason about the internal aspects of the system and refine the requirements. Design allows us to acquire a deeper understanding of issues related to non-functional requirements and restrictions. In design, the system is modeled in such a way

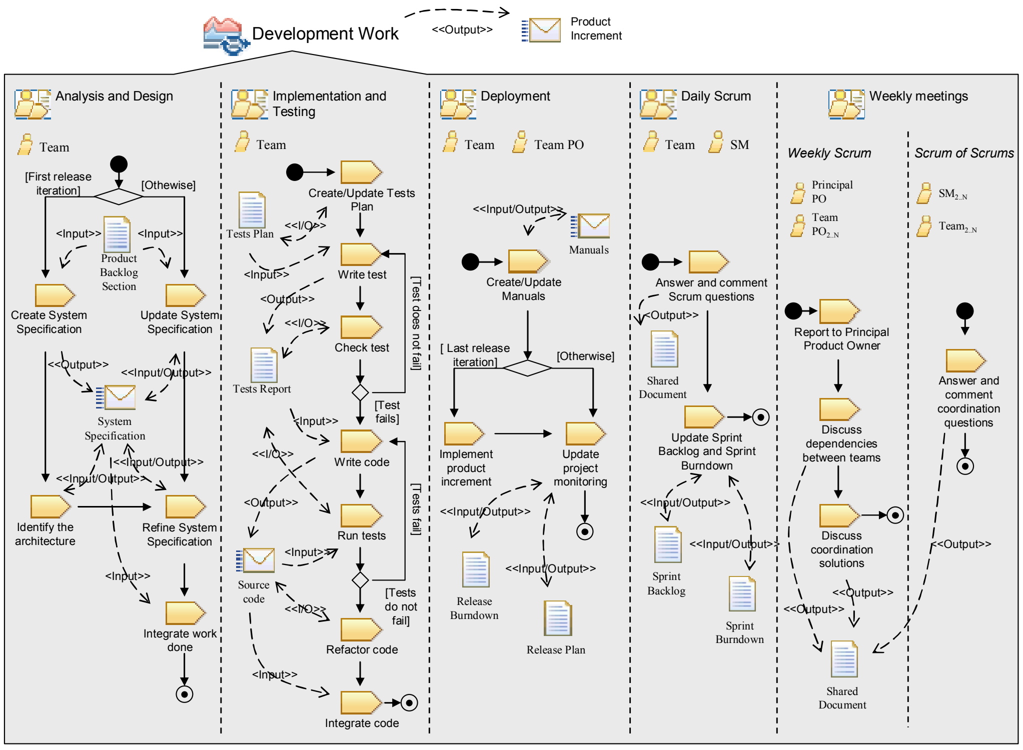


Figure 2. Activity Diagram of the Development Work phase

as to support all the requirements, and an abstraction of the implementation of the system is created. The result of design is the Design Model, which attempts to preserve the structure of the Analysis Model; it serves as a scheme for implementation. In *scRumUP*, implementation and testing are performed jointly, due to the application of the Test-Driven Development practice. Implementation is the center of the construction phase and it transforms the Design Model to executable code that meets user expectations. The most important objective of implementation is to develop the architecture and the system as a whole. In this activity, the use of Continuous Integration (CI) is very important, since code from different teams has to be integrated. There must also be a single repository, supported by version control systems. This activity will focus on the implementation of Sprint Backlog tasks and on monitoring Continuous Integration and Test-Driven Development (see section III.A.3.Practices). The tests, whose purpose is to verify the result of the implementation, should follow a Tests Plan that will be created at the beginning of this activity or in the Release Planning phase; and they will be carried out using Test-Driven Development (TDD) or Acceptance Test-Driven Development (ATDD). ATDD and TDD practices can be considered a form of development, since first of all tests are created and then the code that passes these tests is written. Test scripts and the result of testing will be written up by a document called Test Report, which will be shared by members of all the teams.

The deployment activity will manage all aspects related to the operation on the user's environment. The first task of this activity is to create or update the manuals that will help those who will use the product increment created during the development work. If this is the latest iteration of the release, it will be necessary to implement the product increment in its operating environment. The last task of this activity is to update the project monitoring by the corresponding Team Product Owner.

The next activity is the Daily Scrum. It is a meeting of about fifteen minutes in length, whose purpose is the short-term planning of the sprint for each one of the teams. The ScrumMaster and the team are present at this meeting. Each team member answers the three daily Scrum questions: What did you do yesterday? What are you going to do today? and, Do you have any blockers?

The Weekly Meetings activity consists of two different meetings. On the one hand we have Weekly Scrums, which are the weekly meetings of Team Product Owners and the Principal Product Owner, allowing the latter to track the project and the Team Product Owners to coordinate distributed teams from the point of view of dependency management. On the other hand, we have Scrum of Scrums, which are also weekly coordination meetings but where representatives of different distributed teams are present. At this meeting they discuss dependencies, integration issues, and any other issues that could impact other teams.

The next phase of *scRumUP* is the Sprint Review phase, which aims to ensure that teams have met the goal of the sprint and shows the results to the stakeholders. This phase is divided into three activities: Team Demo, Product Owner Demo and Stakeholders Demo. In the Team Demo activity a demonstration of the work done is made to the Team Product Owner. In the second activity, which is the Product Owner

Demo, each Team Product Owner shows the work done by his team in the sprint. Then the demonstration to be presented to the stakeholders is prepared by identifying which parts of the work done by each of the teams will be displayed, as well as which individuals will attend the meeting. The last meeting is the Demo stakeholders, in which stakeholders, the Principal Product Owner, the Team Product Owners and representatives of the teams are present, and a demonstration of the work done is given to the stakeholders, so that the comments that they can offer on this job are collected.

The last phase of *scRumUP* is the Sprint Retrospective, which aims to analyze the previous sprint to see how it was carried out and to apply the lessons learned in the following sprint. This phase consists of two activities: Team Retrospective and Retrospective of Retrospectives (optional). In the Team Retrospective activity, the team identifies what practices worked well and also what problems arose during that phase, to try to find solutions to prevent these problems from recurring in the next sprint. If there were dependencies between the teams in the previous sprint, an optional meeting called Retrospective of Retrospectives may be held to analyze the sprint from the point of view of coordination between teams and to discuss the problems identified, as well as what can be done to solve these problems.

As for the roles, in *scRumUP* we find the same ones as those found in Scrum:

- The ScrumMaster is responsible for ensuring that the process is understood and followed. The ScrumMaster must be seen as a means of connection between members of the team, not as a team representative. There is a ScrumMaster for each team.
- The Product Owner is responsible for maximizing the value of the work done by the team. In *scRumUP* a hierarchy of Product Owners exists (see Fig. 3). There will be a Principal Product Owner, who will have project management functions and who will provide the overall direction and priorities for the product, as well as several Team Product Owners, who will coordinate the priorities of their Product Backlog team with the other teams.
- The team does the job. It must have the ability to self-organize and to complete a customer-centric feature.

Finally, the artifacts that need to be produced and maintained in *scRumUP* are shown in Table 1. We have defined templates to support each one of these artifacts. In these templates an overview of the artifact is provided, specifying a brief description, its purpose, and its scope, but there are also specific sections for each artifact based on the information that has to be stored for each one.

TABLE I. ARTIFACTS

Name	Description
Requirements Specification	It is a list of all the functional and non-functional requirements identified in the Requirements Elicitation activity, organized by features.
Business Case	It contains the business context, costs, budgets, the goal of the release, factors that indicate whether the goal of the release has been met, a probable delivery date, a list of risks, and the Use Case Model.
Product Roadmap	It is a document that specifies to a high level when a set of features will be delivered.
Product Backlog	It is a prioritized list of product features which the Product Owner owns.
Release Burndown	It is a graph that shows the estimated amount of remaining Product Backlog across the time of a release.
Release Plan	It is a document that aims to detail the planning for releases that are closer in terms of time.
Sprint Backlog	It is a list of tasks with all the work that the team identifies as necessary to achieve the sprint goal, i.e. to turn its section of the Product Backlog into a potentially shippable product increment.
Sprint Burndown	It is a graph that shows the estimated amount of remaining Sprint Backlog across the time of a sprint.
System Specification	It consists of the Analysis Model and the Design Model.
Tests Plan	The Tests Plan should include a test strategy, tools and resources needed for these tests.
Tests Report	This artifact will store the test scripts created in the implementation and testing activities and the results of all the tests performed by the teams.
Source code	The source code produced by different teams during the implementation and testing activities will be stored and documented in this artifact.
Manuals	In consists of three manuals: User Manual, Maintenance Manual and Operation Manual.
Potentially shippable product increment	The product increment is created during the sprint and it is the output of this process.
Shared document (Wiki)	There are three shared documents: The Team's Wiki, Shared Wiki of the Teams, and The Product Owner's Wiki

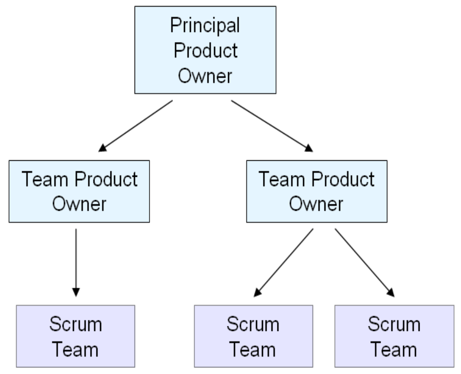


Figure 3. Product Owner Hierarchy

IV. APPLICATION OF THE METHODOLOGY

The methodology presented in the previous section has been adapted to that of a Spanish software company of technology and innovation. The firm is a leader in solutions and services in sectors such as Transport and Traffic, Energy and Industry, Public Administration and Health, Financial Services, Security and Defense, or Telecom and Media. This company operates in over 100 countries and employs more than 30,000 professionals worldwide, sharing its knowledge of different sectors and countries to find innovative solutions to challenges that customers face. It was interested in applying agile methods without having to change its development process completely. The company gave us its methodology so it could be enriched with *scRumUP* to strengthen its distributed development process. The methodology of the company consists of several methodologies, and many of these are applied in parallel during the software development. Since Scrum is a generic framework for project management, the activities of *scRumUP* shown in the previous section are still kept in this adaptation. It has been necessary to add other phases and activities of the company methodology to the process, however. The main criterion used for the adaptation is to maintain the consistency of the methodology. That being so, the activities involved in project planning in the methodology of the company are now in the Release Planning phase, and the verification activities (such as Formal Technical Reviews and Peer Reviews) are at the Sprint Retrospective phase. The three main methodologies of the company are still carried out in parallel, but now, thanks to the activities Daily Scrum and Weekly Scrum, the coordination between the teams and the project tracking is increased.

The different methodologies that existed in the enterprise and that were performed in parallel were: Development Methodology, Measurement Methodology and Testing Methodology. The Development Methodology was responsible for defining the development environment with the roles, responsibilities, tools and work products. The Measurement Methodology defined phases, activities and techniques needed to perform measurements and obtain metrics. Finally, the Testing Methodology defined phases of a test cycle and its alignment with the different phases of the life cycle of a product. A part of the general methodology was responsible for continuous verification, by means of the activities Peer Reviews and Formal Technical Reviews.

After a discussion of possible ways of integrating *scRumUP* with the methodology of this company, it was decided that the best form of integration was the one shown by Fig. 4.

The process is still composed of two distinct parts: the first part is the Release Planning phase, in which the release is planned; and the second is a sprint made up of the following phases: Sprint Planning, Development Phase, Measurement Phase, Testing Phase, Sprint Review and Sprint Retrospective.

Two of the activities related to planning of the company prior methodologies have been added to the Release Planning: Initial Estimate (from the Measurement Methodology) and Planning and Management (from the Testing Methodology). In the Initial Estimate activity a first approximation of the initial estimate template of the measurements is performed. In the Planning and Management activity the testing process is organized, by answering the questions of how much, how, when and by what means, the different tasks of the test group are going to be performed. It was decided to separate these activities from their respective methodologies, to add them to the Release Planning phase, thereby maintaining a consistent methodology. It is at that Release Planning phase, after all, that the initial planning of the release is done.

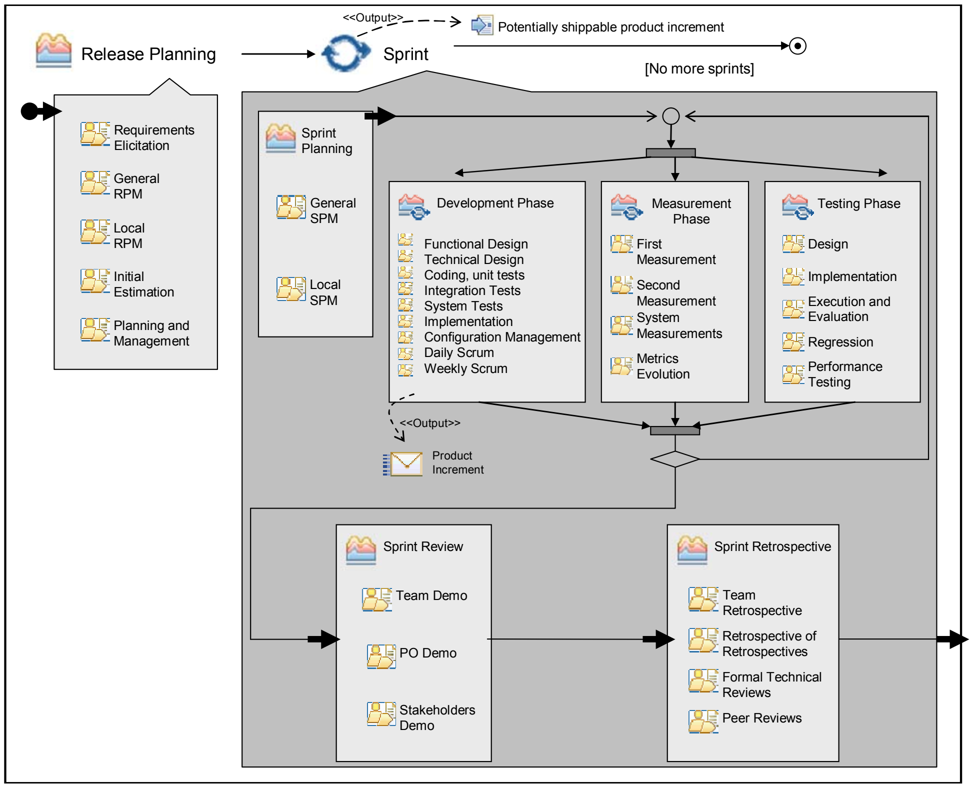


Figure 4. Integration of the methodologies

The Sprint Planning and Sprint Review phases do not change with respect to *scRumUP* .

In this new adaptation, the prior development work now corresponds to the three core methodologies of the company, carried out in parallel: Development Phase, Measurement Phase, and Testing Phase. The Development Phase corresponds to the Development Methodology, along with two additional activities: Daily Scrum and Weekly Meetings. The Measurement and Testing phases correspond to the Measurement and Testing methodologies respectively, but without the initial activities related to planning, which are now in the Release Planning phase.

Finally, the activities for verification of the methodology of the company cited below have been added to the Sprint Retrospective phase: Formal Technical Reviews (FTRs) and Peer Reviews. The consistency of the methodology is thus maintained, since the goal of this phase is to analyze the previous sprint, to see how it went and to apply the lessons learned. The activity Formal Technical Reviews is carried out by each multidisciplinary team, in order to ensure the development of products under the appropriate technical parameters, as well as to detect errors and get feedback from the processes and the development standards. The Peer Reviews activity is carried out between different teams, to perform a series of "peer" checks over the life cycle of projects. They are similar to FTRs, but are carried out among different projects by technical staff from each of the projects that intersect.

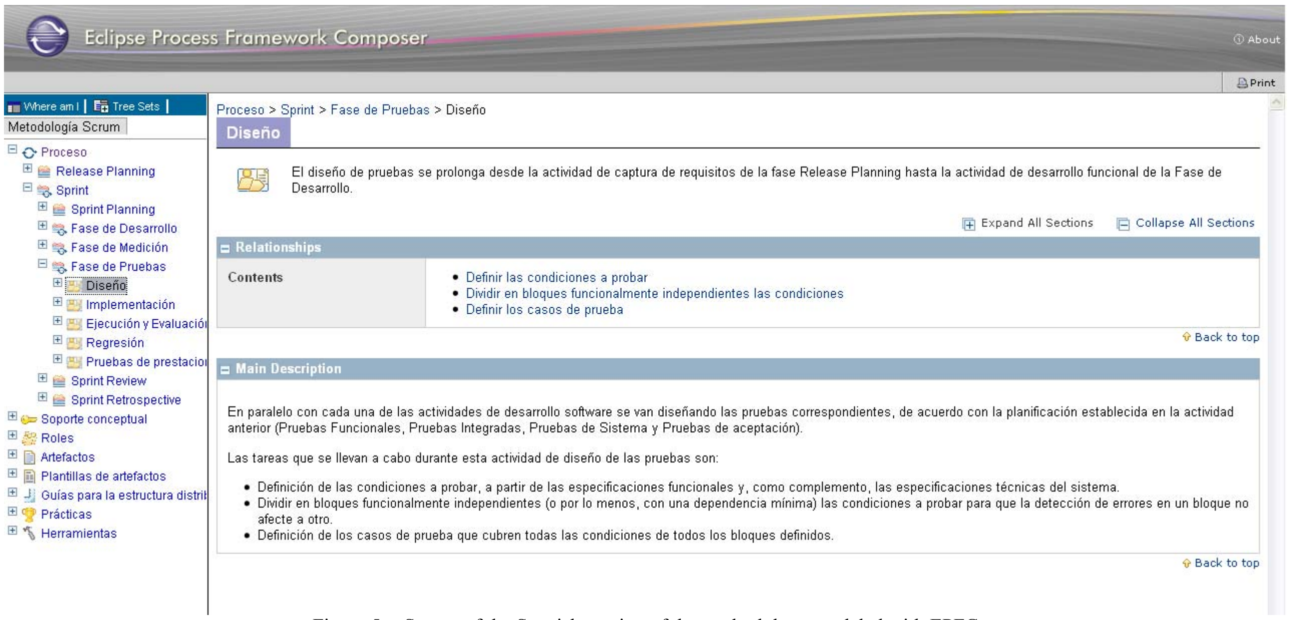
The integration of the methodologies was presented to the company and reviewed by its quality control department. From that moment on, constant communication was maintained with members of the quality department. The aim was to refine the methodology and to adapt it to the needs of the company. After some modifications, the company finally accepted the methodology, and since then this methodology is being applied in a pilot project.

The methodology was modeled with the Eclipse Process Framework Composer tool (Fig. 5), in order to be available for all of the distributed teams easily and quickly. Employing this tool, the activities, tasks, roles, work products and guidelines needed to carry out the distributed process are described and made available on the web.

V. Conclusions and Future Work

In this article we have proposed *scRumUP,* a methodology for distributed software development based on a combination of one of the most widely-used agile methods, Scrum, with one of the most established development methodologies, the Rational Unified Process (RUP). It is intended to cover the need for a hybrid proposal for distributed development, uniting the benefits provided by agile methods with the robustness of the Rational Unified Process (RUP). This methodology can be beneficial for those organizations that perform software development in a distributed way, since it takes into account many of the best agile practices and it has been specifically developed for distributed environments.

The application of agile methods like Scrum to distributed software development can be beneficial, since these methods emphasize the most critical challenges of distributed development, such as communication and coordination. The application of RUP to distributed software development strengthens it by means of an established development process, and by ensuring that there is the documentation needed to enable project monitoring for distributed teams. The fact that both methodologies, Scrum and RUP, are generic frameworks that allow adaptation to different needs has made the integration of both methods less complicated in order to create *scRumUP*. The agile practices Test-Driven Development (TDD) and Continuous Integration (CI) have also been integrated in the methodology. The use of TDD is important, because it improves the design and produces higher- quality code [15]. The application of CI is justified, as several teams



Proceso > Sprint > Fase de Pruebas > Diseño

Diseño

El diseño de pruebas se prolonga desde la actividad de captura de requisitos de la fase Release Planning hasta la actividad de desarrollo funcional de la Fase de Desarrollo.

Relationships

Contents
- Definir las condiciones a probar
- Dividir en bloques funcionalmente independientes las condiciones
- Definir los casos de prueba

Main Description

En paralelo con cada una de las actividades de desarrollo software se van diseñando las pruebas correspondientes, de acuerdo con la planificación establecida en la actividad anterior (Pruebas Funcionales, Pruebas Integradas, Pruebas de Sistema y Pruebas de aceptación).

Las tareas que se llevan a cabo durante esta actividad de diseño de las pruebas son:

- Definición de las condiciones a probar, a partir de las especificaciones funcionales y, como complemento, las especificaciones técnicas del sistema.
- Dividir en bloques funcionalmente independientes (o por lo menos, con una dependencia mínima) las condiciones a probar para que la detección de errores en un bloque no afecte a otro.
- Definición de los casos de prueba que cubren todas las condiciones de todos los bloques definidos.

Figure 5. Screen of the Spanish version of the methodology modeled with EPFC

may be working on the same part of a product if the development is distributed, and therefore a common integration of the whole code is needed if we are to detect errors early. The RUP approach for implementation and testing has not been used, since the development proposed by RUP is based on components (prebuilt objects with well defined interfaces that can be used in different places), and Scrum agile development is based on features (units of functionality from the user point of view). That is the reason why the integration of these practices with RUP has been less straightforward. Moreover, TDD and CI are more appropriate for supporting distributed development, as they allow continuous testing and code integration.

We are now awaiting the results of the first pilot project, expecting them to provide more information about the performance and the utility of the methodology. When the company provides us information of the application of the methodology we will present an experience report with details about the project and the distribution. Our future work is therefore to improve further the methodology from the feedback we will have obtained from the results of the pilot projects carried out within the company. We also want to apply *scRumUP* in other companies to get feedback with a broader view in order to consider the contexts in which the methodology can apply. The purpose of that would be to provide a thorough methodology that can be applied well in distributed environments.

Acknowledgments

We acknowledge the assistance of the following projects: ORIGIN (IDI-2010043(1-5)) funded by CDTI and FEDER, PEGASO-MAGO (TIN2009-13718-C02-01, FEDER and MEC of Spain) and Agreement Unicauca-UCLM (4982-4901273). Francisco J. Pino acknowledges the contribution of the University of Cauca, where he works as Associate Professor.

References

[1] Ambler, S.W., The Distributed Agile Team. Dr. Dobb's Journal, 2009. 34(1): p. 45-47.G.

[2] Batra, D., Modified agile practices for outsourced software projects. Communications of the ACM, 2009. 52(9).

[3] Kussmaul, C., R. Jack, and B. Sponsler, Outsourcing and offshoring with agility: A case study. 2004. p. 147-154.

[4] One-Ki (Daniel) Lee, Probir Banerjee, Kai H. Lim, Kuldeep Kumar, Jos van Hillegersberg, and Kwok Kee Wei. 2006. Aligning IT components to achieve agility in globally distributed system development. Commun. ACM 49, 10 (October 2006), 48-54.

[5] Lee, S. and H.S. Yong, Distributed agile: project management in a global environment. Empirical Software Engineering, 2009: p. 1-14.

[6] Phalnikar, R., V.S. Deshpande, and S.D. Joshi. Applying agile principles for distributed software development. 2009. Singapore.

[7] Jeff Sutherland; Anton Viktorov; Jack Blount; Nikolai Puntikov; , "Distributed Scrum: Agile Project Management with Outsourced Development Teams," *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on* , vol., no., pp.274a, Jan. 2007.

[8] Schwaber, K. and J. Sutherland. Scrum Guide. 2010; Available from: http://www.scrum.org/storage/scrumguides/Scrum%20Guide.pdf.

[9] Jacobson, I., G. Booch, and J. Rumbaugh, El Proceso Unificado de Desarrollo Software. 2005: Addison Wesley. 438.

[10] Larman, C. and B. Vodde, Practices for Scaling Lean & Agile Development. 2010: Pearson Education.

[11] Woodward, E., S. Surdek, and M. Ganis, A Practical Guide to Distributed Scrum. 2010: IBM Press.

[12] Sutherland, J.; Schoonheim, G.; Rijk, M.; , "Fully Distributed Scrum: Replicating Local Productivity and Quality with Offshore Teams," *System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on* , vol., no., pp.1-8, 5-8 Jan. 2009.

[13] Fowler, M. Continuous Integration. 2006; Available from: http://www.martinfowler.com/articles/continuousIntegration.html.

[14] Larman, C. and B. Vodde, Scaling Lean & Agile Development. 2009: Pearson Education.

[15] Sangwan, R.S. and P.A. Laplante, Test-driven development in large projects. IT Professional, 2006. 8(5): p. 25-29.