RAMBUS: An Agile Process for Developing Web Applications

Vinícius Pereira, Antonio F. do Prado
Department of Computing
Federal University of Sao Carlos - UFSCar
Sao Carlos – São Paulo
Brazil
{vinicius_pereira, prado}@dc.ufscar.br

ABSTRACT: *In this paper is described an agile process for Web development based on the use of User Stories. The main objective of this study is to have a greater integration between the disciplines of software development process, i.e. to have a closer relationship between requirements and the final product, and all stages between these two extremes. In addition, the process incorporates the use of acceptance tests. The User Stories try to be a snapshot as faithful as possible of User's needs. It will be used both in manuscript form (in cards) for requirements specification and in an executable format to guide the coding and serve as acceptance tests. Other artifacts, such as Class Diagram, Architecture Model and Product Backlog also will be created.*

Keywords: User Story, User Behavior, Agile Development, Web Applications

Received: 12 December 2010, Revised 18 January 2011, Accepted 25 January 2011

1. Introduction

In recent years, changes in the interactions between users and companies with the Web have made the delivery of services and interactions between users became more dynamic and accessible leading to the appearance of the term Web 2.0 [1]. This term refers to a new generation of Web applications designed specifically to support the sharing of user-generated content.

In the same period agile methods have begun to expand its use in the job market. The companies sought cheaper ways to develop their applications and improve their work environment. This was possible with the advent of agile methodologies. Thus, nowadays it is common to find companies that use the main agile methods: Scrum [2] and eXtreme Programming (XP) [3] or even a combination of both.

A difficulty in Software Engineering is to integrate effectively the steps of the development process. The artifacts generated at each step are not necessarily used in the later stages as a result of too much documentation that makes it difficult to use, resulting in wasted time in its creation, and the loss of information relevant to the process.

It will be presented in this paper an agile process aimed at to deal justly with the dynamism of the Web environment and its social applications [4]. The general objectives of the proposed process are: (1) deliver a documentation involving the whole process, and (2) integrate to the process the use of acceptance and functional tests, and indicate when unit tests can be used. Thus the main goal of the process is to facilitate the evolution and maintenance of Web application.

2. Concepts and Techniques

For this study it was analyzed the different concepts and techniques found in the traditional Software Engineering. Among

them it is highlighted the concept of User Story [5]. In addition further research on key agile methods - with emphasis on Scrum and XP - it was performed to obtain better knowledge of them.

To integrate the documentation, the process makes use of User Stories in the form of cards in conjunction with the Navigation Model to specify the requirements. It was analyzed a way to use the Navigation Model [6] to help in the transcription of these cards for an executable version in order to guide the coding, thus avoiding misinterpretation of requirements. To reach this stage was also seen how analyze and design the application based on this same material. In the end of the process all the disciplines must have some version of User Stories (card or executable) as one of its major artifacts, thus allowing greater integration between them.

2.1 User Story

The User Story describes functionally what is required and valuable to the user. In a User Story there is three C's that are: Card, Conversation and Confirmation; and follows the principle INVEST: Independent, Negotiable, Valuable for the user, Estimable, Small and Testable. The Card, known as Story Card, is the User Story written and formalized. Through the use of this the Conversation can be restored in order to get Confirmation.

The idea of an User Story be written on a card and not in other media is due to the principle Small (to be short). However if the User Story exceed the limit of the card, it can be targeted. There is no limit to the amount of User Stories, since they are the pattern set by Mike Cohn [5]. An informal example to illustrate the pattern is: "Students can purchase parking passes". In his book, Mike Cohn suggests a more formal approach to write User Stories [5]: As a "paper" I want "something" so that "benefit". This approach facilitates the determination of why a feature should be built and for whom, so this is the approach commonly used. The same example would look like this: "As a student I want to purchase a parking pass so that I can drive to school".

Agile methods promote interpersonal communication rather than an extensive documentation and quickly adapt to change instead of worry about reporting the problem excessively. Using the User Stories this goal can be obtain because they: (1) represent small pieces of the business value that can be implemented in a period of days or weeks, (2) requires little maintenance, (3) allows developer and user to discuss the requirements throughout the project's lifetime, (4) enables the segmentation of the project in small increments, (5) are suitable for projects where the requirements are dynamic, such as Web 2.0, or difficult to understand, and (6) requires close contact with the user throughout the project so that the parts of higher value business application are implemented correctly.

2.2 Scrum

Scrum is a framework for agile development of software that is interactive and incremental. It was initially designed as a way of project management for the automotive and consumer industries. These sectors have noted that by using small multidisciplinary teams (cross-functional) in their projects the final result was considerably better. Ken Schwaber [2] formalized the definition of Scrum and helped to introduce the concept in the development of applications around the world. The Scrum has three main roles: (1) the ScrumMaster, who maintains the processes, (2) the Product Owner (PO), which represents stakeholders and business, (3) the ScrumTeam, a cross-functional group composed of about seven people who do analysis, design, implementation, testing and other tasks of application development.

During each Sprint, typically a period of two to four weeks, the ScrumTeam creates an increment of product that can be put into production. The resources that go into a Sprint came from the Product Backlog. This, created by the PO, is a set of prioritized requirements by the User. The items that are in the Product Backlog which will go to the Sprint are determined during the Sprint Planning Meeting. During this meeting the PO tells the ScrumTeam about items in the Product Backlog with the highest priority to the User. Then the ScrumTeam determines how many items they can be committed to complete during the next Sprint.

Once the Sprint has begun no one is allowed to change its content, this means that the requirements of that Sprint are unchanged. The development has a fixed deadline so that the Sprint must end on time. Every day, the ScrumTeam came together in the Daily Scrum Meeting which is a quick meeting on foot to expose what is being done and what the difficulties encountered. It is up to the ScrumMaster to resolve or minimize these difficulties. If any requirement is not completed for any reason during this period, it is left out and back to the Product Backlog. At the end of a Sprint, the ScrumTeam meets to

analyze its positive and negative points (Sprint Retrospective). It also shows the resulting application to the User (Sprint Review).

3. The RAMBUS Process

The RAMBUS (Rambus Agile Methodology Based on User Stories) is an iterative and incremental process for developing Web applications composed of four phases - Requirements, Planning, Executing and Closure - and three disciplines - Communication, Modeling and Construction. With the help of the Navigation Model, RAMBUS makes use of User Stories to get the initial requirements and keep the entire development process. At the end of each iteration - treated as mini-projects - are developed functional versions of the application, based on User Stories used which over time will include all the features requested. It is used the theories of Behavior Driven Development (BDD) [7], through the stories that test the user's behavior, and established concepts of Scrum, including Product Backlog and Sprint. Also it is suggest the use of some good practices such as Pair Programming, Test Driven Development (TDD) [8] and Continuous Integration.

The process runs in cycles to create the Web application requested. The Figure 1 is presented the life cycle of the RAMBUS process in the form known as hump chart. Vertically are the disciplines of the process proposed in this study (Communication, Modeling and Construction) and horizontal, represented by the Scrum elements to assist in understanding, are the phases (Requirements, Planning, Execution and Closure).

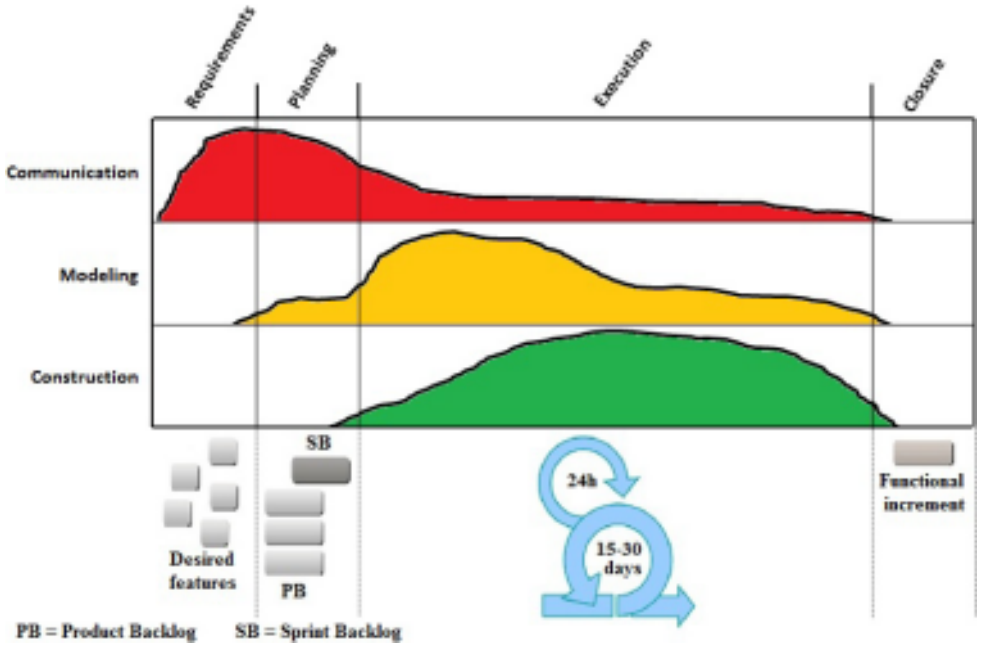

Figure 1. Life cycle of the RAMBUS process

It is observed from the Figure 1 that the disciplines have their usage spikes at certain phases of the process. This means, for example, that while the requirements specification is made, the team is in the Requirements phase, with greater focus on Communication discipline. When it is created the Product Backlog and selected what will be done in the iteration, the team is in the Planning phase, but still focused on Communication. In Execution, the development team is in a cycle as the Sprint of Scrum, with greater focus, initially, in the discipline of Modeling and then the Construction discipline. And finally, the Closure phase includes the delivery of increment functional created in the iteration and process analysis for improvements in the next iterations. As shown in Figure 1, once started a discipline it continues active with a lower intensity until the last phase of iteration. For example, despite the discipline of Communication be more focused on the Requirements and Planning phases, it is still there in the Execution phase, but with less intensity.

To complement the understanding of the RAMBUS in the Figure 2 is shown in a SADT (Structured Analysis and Design Technique) diagram [9], a more detailed version of the process disciplines with their respective artifacts designed to aid the process.

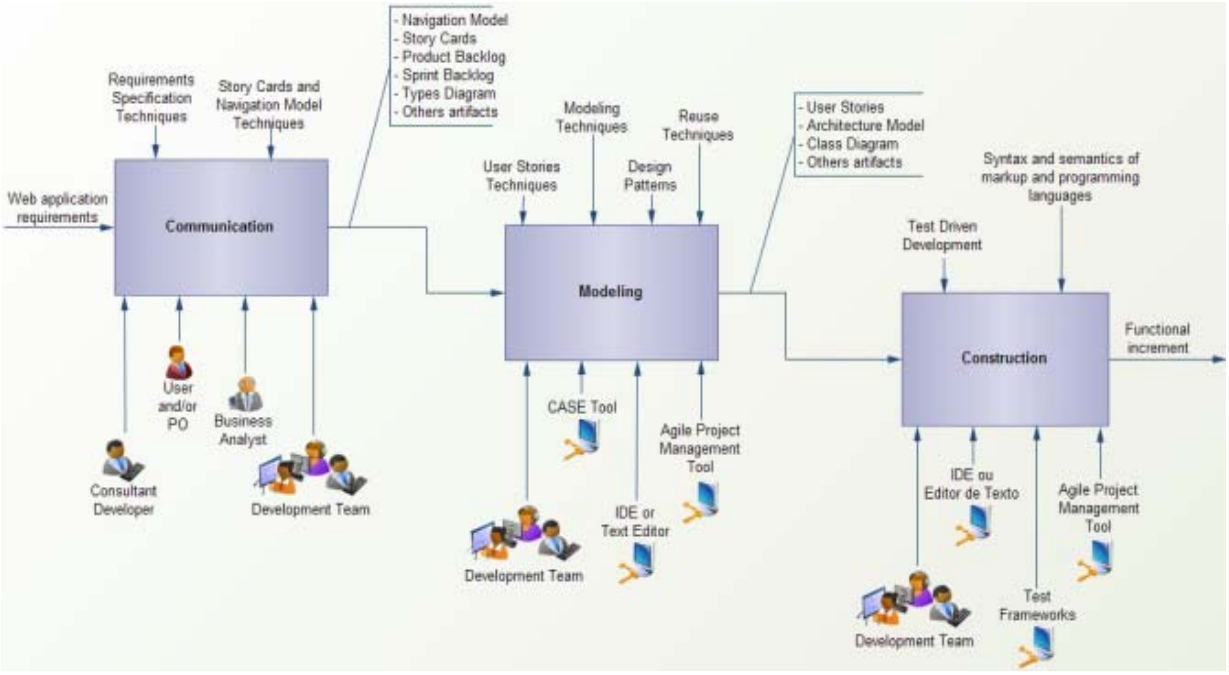

Figure 2. The Disciplines of the RAMBUS Process and its Artifacts

Some features may only be requested when the user interacts with the application. This fact will generate changes in the project and the team must prepare the system to evolve with the minimum difficulty possible. Therefore, throughout the process, it is worth testing ideas and collect user's feedback, because the early detection of problems can save time and resources spent to repair them. Thus, this type of development involves a lot of changes throughout the project.

In the next sections it will cover the disciplines with their artifacts and later the phases of the process. In addition, to assist in the presentation is used as example of Web application the wiki, a type of social application - called groupware or collaborative software. Therefore, all figures show examples of specifying a wiki to illustrate the description of artifacts.

3.1 The process disciplines

This section describes the four disciplines of the process. It will be shown in details the main artifacts of each phase besides how to create and use them.

Communication: In this discipline, the focus is the interaction between people involved in the process, especially for the requirements specification and elaboration of which will be held at Sprint.

To assist in the requirements specification, the process proposed here makes use of Story Cards and Navigation Model. There is no rule about which of the two artifacts must be created first, because the process assumes that one complements the other. It is noteworthy that throughout the study, the term Story Card expresses a User Story in the card format proposed by Mike Cohn [5]. At the end of the conversation with the user the Story Cards and the Navigation Model should be ready so that together they can display an overview of the application to be developed in the next discipline.

For presentation purposes the Story Card will be the first artifact with explanations of its use. The Figure 3 shows the front of a Story Card requesting the management of pages in a wiki.

As the Figure 3 shows, the Story Card is simple and objective. It allows the user to have a greater understanding of the process and thus collaborate in the best way to identify the requirements. However, collecting data only in this way is not enough. This information has a very high level of abstraction, even when considering the back of the card - which will be shown later. It is noteworthy that the card contained its priority to the user and an initial estimate of the difficulty to perform the required functionality.

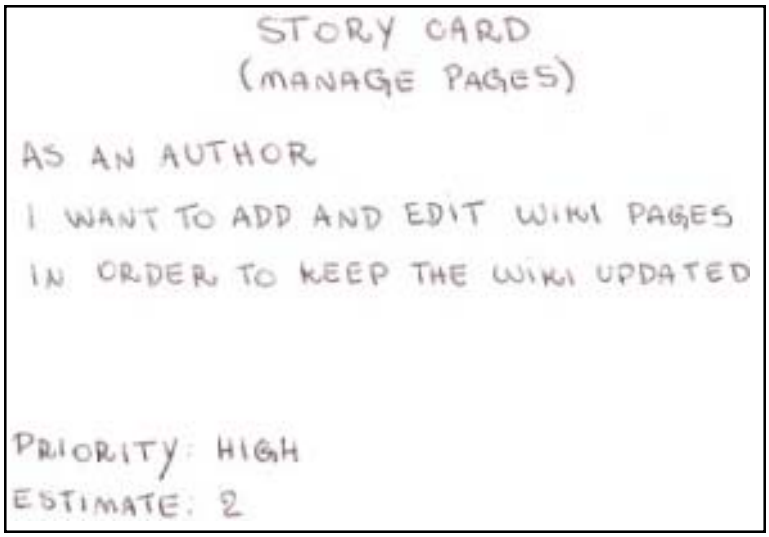

Figure 3. Example of Story Card to the wiki

Therefore the user say "what" expects the application should do, but not "how" it should function. This is the point where it is necessary to use the Navigation Model. As the name suggests, this artifact shows clearly how will be the data flows in the Web application. The Model should only contain vital data both for its understanding - such as identifying the page name, direction and content - as for a Story Card (or a group of them) has its flow easily identified in the Model.

Since this discipline requires a close contact with the user, the more informal the approach is - without compromising the credibility of the work - the greater the chance to encourage him to interact. Thus, drawing this Model with pen and paper, as shown in Figure 4, or even a blackboard is an efficient way to achieve that result.

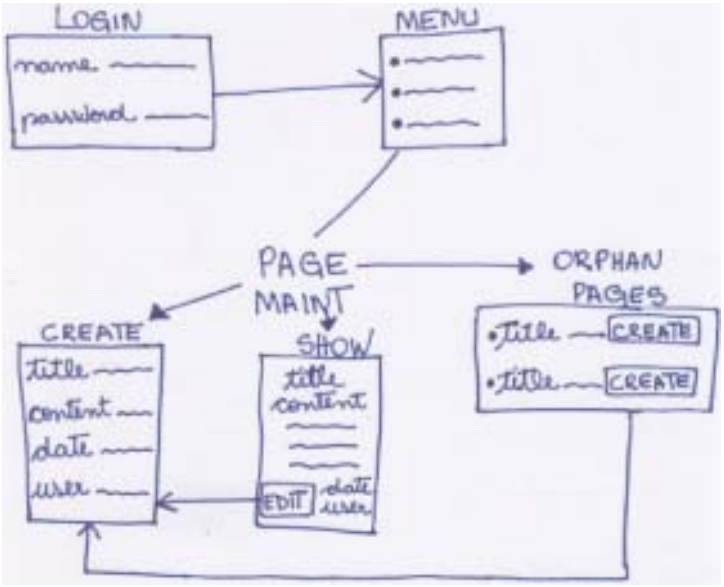

Figure 4. Example of the Navigation Model to the wiki

By merging the two artifacts the requirements become more detailed during the conversation. This is the main gain in the area of specifying requirements. Business rules that are not explicit in the Navigation Model, or even to make them clearer, can be written on the back of the correspond Story Cards as shown in the Figure 5.

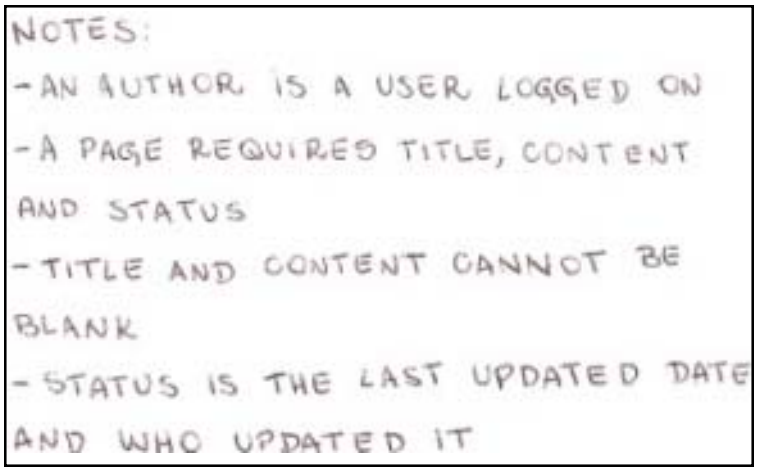

Figure 5. Back of the Story Card in the Figure 3

Nothing prevents other artifacts from to be created in the search to express the requirements more clearly. The more practical the artifact is, without loss of content, the greater the likelihood that the user interacts. An example is the Types Diagram [10], which is extremely simple, using only the entities (with the possible attribute names) and their relationships. The user does not need to know what an entity is, just understand what the drawing represents and interact. The Types Diagram can also be used to specify the information on a legacy database that can be used by the Web application. The Figure 6 shows a Types Diagram to illustrate this example.

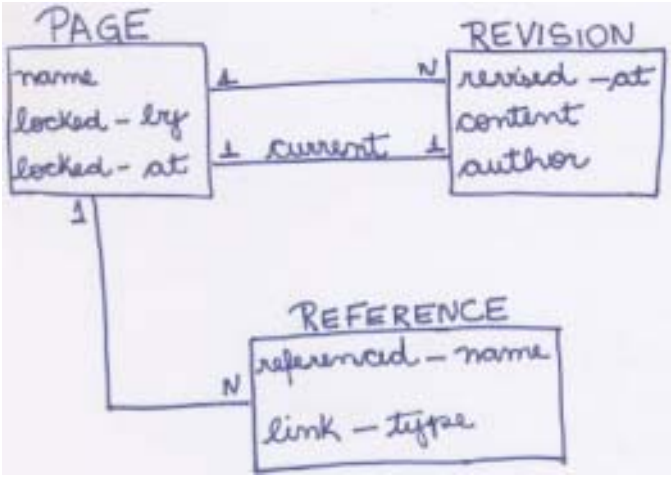

Figure 6. Example of a Types Diagram for the wiki

Technical issues such as infrastructure are not discussed in Story Cards but should be recorded for later analyzed by the development team. If these issues have a direct relationship with a feature (or more) is important to note which Story Card(s) they are related.

By using the Story Cards and the Navigation Model a person with the role of Product Owner (PO) must create a list of desired features and prioritize them according to the needs required by the user. This list is the Product Backlog known of Scrum. Therefore during the iteration planning meeting the PO presents to the development team the features prioritized and their descriptions. Thus, there are specific which items of the list will be made in the current iteration, creating the Sprint Backlog - similar to what is done at the meeting prior to the Sprint in Scrum.

Also, the development team must verify the existence of Activities required for a feature (or a group of them) be implemented correctly. Such Activities may be present in Product Backlog and must be present in Sprint Backlog. Nothing prevents the team to segment those features (and their activities) into smaller tasks among its members, in order to get a better development

process. To better control the Backlogs, a good alternative is to use a tool for agile project management. Another recommendation is to print the items of the current iteration and put them in a place for easy viewing for everyone involved, in the KanBan style [11].

At the end of Communication at least these artifacts (as seen in Figure 2) must have been prepared: the Story Cards and the Navigation Model, both are essential to guide the construction of artifacts of the next discipline; and the Product Backlog and Sprint Backlog to control the desired features and what are being done in the moment.

Modeling: In this discipline, the development team makes use of Story Cards, Navigation Model and any other artifacts that were created in Communication to help in the requirements specification to perform the analysis and design of the Web application. To support these activities in this discipline other artifacts are generated or even redefined in the case of changes. The development team should prepare a Class Diagram or refine a previously created Types Diagram which will provide assistance for data analysis of the application. This Class Diagram, possibly simple at first, is refined in each iteration. The team must verify, for example, the need to create an abstract class or an interface to handle the data. It is up to the team using the Class Diagram to create the database (if it is not legacy) or create a Database Model. The goal is to create the database gradually according to the needs of the application at the time.

The next step is to prepare the User Stories based on the artifacts that have been used until now. These stories are written in a language proposed by Dan North [7] - using keywords that will be useful throughout the process, such as a DSL (Domain-Specific Language) - which allows a simpler way to communicate with the user and obtain their understanding. The user can check the User Stories written in this form and approve them or not. These Stories should accurately represent what was expressed by the user in the Story Cards and Navigation Model. In the Figure 7 is shown a transcript of a User Story based on the Navigation Model of the Figure 4, in the Story Card of the Figure 3 and his verse in Figure 5.

```
 1 Feature: Manage pages in the wiki
 2     In order to keep the wiki updated
 3     As an author
 4     I want to add and edit pages
 5
 6     Scenario: Create a page
 7         Given I am on the wiki creation page
 8          When I fill in "Title" with "Test - create page"
 9           And I fill in "Content" with "Checking the content..."
10           And I press "Save"
11          Then I should see the new page
12           And I should see the revision date
13           And I should see the author`s name
14
15     Scenario: Update a page
16         Given that I have created a page "First wiki page"
17          When I press "Edit"
18           And I fill in "Content" with "Updating the content..."
19           And I press "Update"
20          Then I should see the updated page
21           And I should see the new revision date
22           And I should see the authors name
```

Figure 7. Example of a User Story transcribed

This description also provides a “hidden” element: the Scenario. It is a perspective of using of the User Story that contains one or more acceptance criteria. For example, in line 6 of Figure 7, the scenario which is tested refers to the creation of a page. Therefore, all that lies below this line is part of that context, until a new scenario (line 15). A User Story is complete when all the acceptance criteria are fulfilled for each Scenario.

Despite the example of Figure 7 it is important to note that a way to improve the use of User Stories is to create them only to the business rules. Once the development team has set a Class Diagram, the CRUD [12] can be generated automatically from

various CASE tools. So there is the possibility of saving development time, not only in coding but also in maintaining of this automatic code.

Analyzing the User Stories and other artifacts, the team must look for what can be reused from previous applications or what can give rise to an artifact reusable in future applications of this domain. Moreover, it is valid to identify where a pattern can be included to facilitate understanding of the application and subsequent maintenance. An example of pattern is to use the MVC [13], to better separate the layers between the data and user interface, something that helps both the creation and maintenance of a Web application.

In the next discipline, the example will use a framework that generate error messages and warnings self-explanatory, which makes more sense on a project with the code divided in layers (MVC). Therefore the team must choose which framework use to execute the User Stories created in this discipline. Some options are: easyb for Groovy, Jasmine for JavaScript, Cucumber for Ruby, RSpec for Ruby and JBehave for Java.

Also in this discipline, should be choose which hardware and software will be adopted for the project. For example, it can be define the Ruby programming language as a base with Rails framework to assist in the development of Web application. It is therefore necessary to create an Architectural Model to show, for example, how MVC works Ruby and Rails. The Figure 8 shows that. Another decision is which DBMS (DataBase Management System) will be used if necessary to create a database

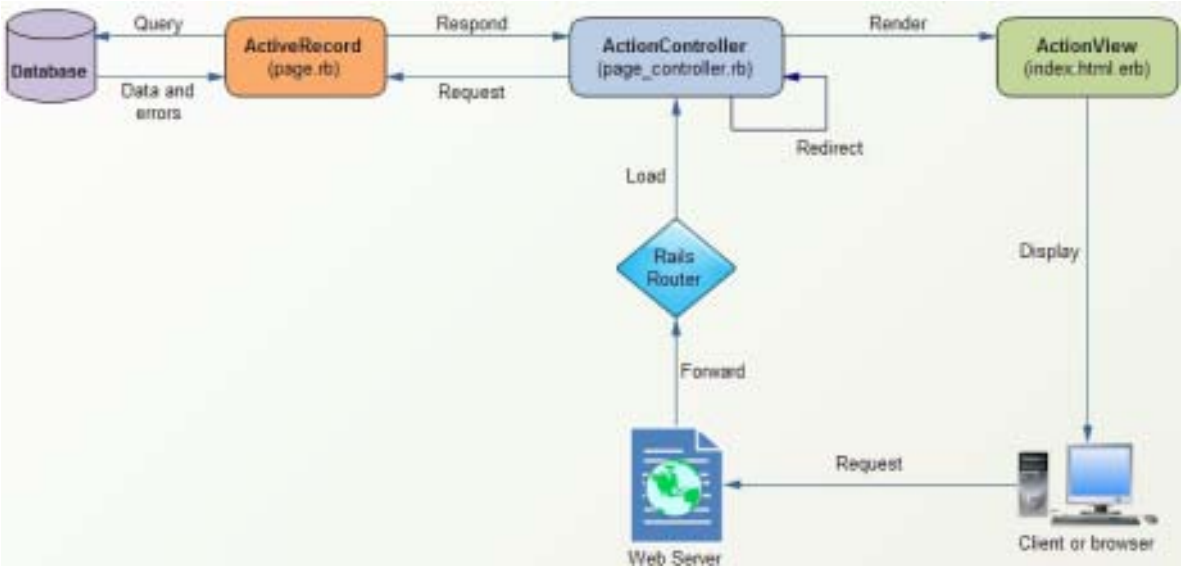

Figure 8. Example of a MVC Architecture Model using the Rails

Thus, at the end of this discipline, as shown in Figure 2 the artifacts created are: the User Stories in executable form, the Class Diagram and Architecture Model. However, nothing prevents other artifacts be created so that the team can understand the problem better.

Construction: This discipline covers the coding and testing. Practices such as Daily Meetings, Pair Programming, Continuous Integration and Pattern Codes are very useful in this discipline as well as any others that help the team to implement the application as best as possible.

Based on the artifacts of previous discipline, the development team must create the necessary infrastructure to support the

[1]http://www.easyb.org/

[2]http://pivotal.github.com/jasmine/

[3]http://cukes.info/

[4]http://relishapp.com/rspec

[5]http://jbehave.org/

Sprint. Probably the first Sprint will have more support activities like to create the database based on the Class Diagram or the Database Model. It is recommended to take this moment to generate all possible automatic codes (like CRUD). Then write and test the code concerning the business rules.

The team must run the User Stories present in the Sprint Backlog and start coding based on the error messages that the chosen test framework returns. The Figure 9 shows the probable first error in the execution of User Story of Figure 7. The Figure 9 shows why it has been adopted a framework for the User Stories written in Modeling. They eventually serve as acceptance tests simulating the behavior that the user wants. It is important to realize that correcting the errors that appear during these tests the development team can create a more faithful application to user expectations, particularly with respect to the processing of business rules.

```
(::) failed steps (::)

Can't find mapping from "the wiki creation page" to a path.
Now, go and add a mapping in /home/vinicius/Projects/wiki/features/support/paths.rb (RuntimeError)
./features/support/paths.rb:27:in `rescue in path_to'
./features/support/paths.rb:21:in `path_to'
./features/step_definitions/web_steps.rb:20:in `/^(?:|I )am on (.+)$/'
features/manage_pages.feature:7:in `Given I am on the wiki creation page'

Failing Scenarios:
cucumber features/manage_pages.feature:6 # Scenario: Create a page

2 scenarios (1 failed, 1 undefined)
14 steps (1 failed, 6 skipped, 7 undefined)
0m0.895s
```

Figure 9. First error when running the User Story of Figure 7

In the example of error shown in Figure 9 the simulation of use returns an problem in the first acceptance criteria "Given I am on the wiki page creation." Given is a keyword and the message states that the regular expression "I am on" has an action called by the chosen framework to run the User Stories - in this case, the action is in a file called web_steps.rb. The framework suggests that the rest of the expression - "the wiki page creation" - be mapped. After that, the next message will be about an attempt to fill the Title field with the content expressed in the User Story. If the field do not exists this will cause an error. Next, if the field cannot be filled by a string other error occurs.

This process is repeated iteratively, through tests for all acceptance criteria for each scenario present in the User Stories. The code required is implemented until all User Stories of the iteration do not have any errors. At the end of this, the Sprint finish and a fully functional prototype is ready.

3.2 The process phases

The development process is divided into iterations, each iteration is divided into four consecutive phases. So at the end of the last phase the iteration ends and a new iteration begin in its first phase. Each phase is composed of its purpose, its activities and its generated artifacts.

Requirements: This phase aims to define the objectives and if the project is workable. People involved in this phase are: Consultant Developer, responsible for requirements specification and the interview with the user; the User, who describes the need for the requested Web application and its features; and the Business Analyst, a person with knowledgeable in the area of business of the user that assists the Consultant in the search for requirements and the interview. Nothing prevents the Consultant Developer has knowledge in the business area and so adds the functions of the Business Analyst.

To the Consultant Developer is suggested that specifies the information about the Web application requested by the user in conversations during the project. They will be vital to the development team to analyze and project what will be done in each iteration, estimating time and effort. The requirements specified in this information will be use in the next phase.

The activities required in this phase involve the identification of different types of users who will interact with the application and how such interaction occurs. This will be accomplished in conversations among stakeholders through the use of Story Cards and Navigation Model.

Planning: The purpose of this phase is to analyze the problem domain, develop the project plan, establish the architectural foundation and eliminate the risk elements. These are the risks of requirements, technological - referring to the ability of the available tools - and skills - of the members of the project. People involved in this phase are the development team and the Product Owner (PO).

It should be prepared a Product Backlog with the desired features for the Web application. The PO gets the responsibility of creating the Product Backlog and manages it. The Product Backlog should be presented for the development team at a meeting that will define what will be done in the cycle of coding and testing (like the Sprint of Scrum). The team should analyze this Backlog, select what will make in the Sprint and divide the features into tasks to be executed during the cycle. This list of tasks to be performed is the Sprint Backlog.

It is recommended that a development cycle does not take more than a few weeks [14]. Considering the Scrum guidelines in relation to the Sprint, this "few weeks" can be refined for a period of two to four weeks. All items selected to participate in the iteration must be completed during this period. This is a commitment from the development team and its non-compliance can lead to loss of confidence by the user.

The user can request changes before the start of the Sprint or of items that are not part of it. The control of Backlogs must be constantly updated and, most importantly, the user should be aware of the progress of the project at all time. Beyond that, the development team must foresee an architecture that supports the application which will be developed through an architectural model, where the use of MVC [13] can be shown, for example. To assist in this step is indicated sketching a Types Diagram [10] to solve possible doubts about the application between the team.
For documentation, the indicated is to use a graphical tool or CASE, and store the different versions of artifacts - in relation to the cycles of iterations. However, due to its great dynamics, the teams meetings and conversations with the User require sketches by the convenience and speed they provide.

Execution: The objective here is to develop all the components needed to create a version of the product based on features of the current iteration - without losing what has been done in previous iterations - and respecting the artifacts developed in previous phases.

By representing the implementation of the Web application, this phase can be considered equivalent to the Sprint of the Scrum. However, this process goes into some detail about how the application can be made in search of a closest result of user requests. To achieve these objectives, the activities involve a full understanding of the requirements identified in artifacts in the previous phase and based on them create new artifacts to aid the understanding of the project, as the Class Diagram - or refine the Types Diagram previously created.

Another artifact that must be generated is the User Story following what was seen in Modeling discipline. This is a faithful transcription of one (or more) functionality seen in a section of the Navigation Model and in the respective Story Cards. Therefore, all the features of the Sprint Backlog should be transcribed to the User Stories format which will be useful during this phase. It is advisable to use a project management tool, preferably with the function to deal with User Stories - as shown in Figure 10 - separating them between accepted and not accepted, present in the current iteration and waiting for future iterations.

It is noteworthy that at this point is possible and indicated making use of TDD [8] by creating tests before the application code. Thus, the probability of the final product have errors shall be smaller. The tests can also help the programmer to better understand the "problem" and solve it in a more efficient way. This makes it possible to generate a clean code which according to Robert Martin [15] is "a code that reflects exactly what it was designed to do, without artifice or obscurity".

Regardless of use or not of TDD, the process makes use of tests guided toward behaviors. This is due to the fact that User Stories created in the previous phase are executed as acceptance tests and guide the coding. It is important to realize that this logic is the core of this process. All artifacts generated so far aims to guide the coding based on the expected behavior that

the Web application should have.

Closure: Finally, the purpose here is to present the product created in the iteration to the user and put it into production, if that is his will.

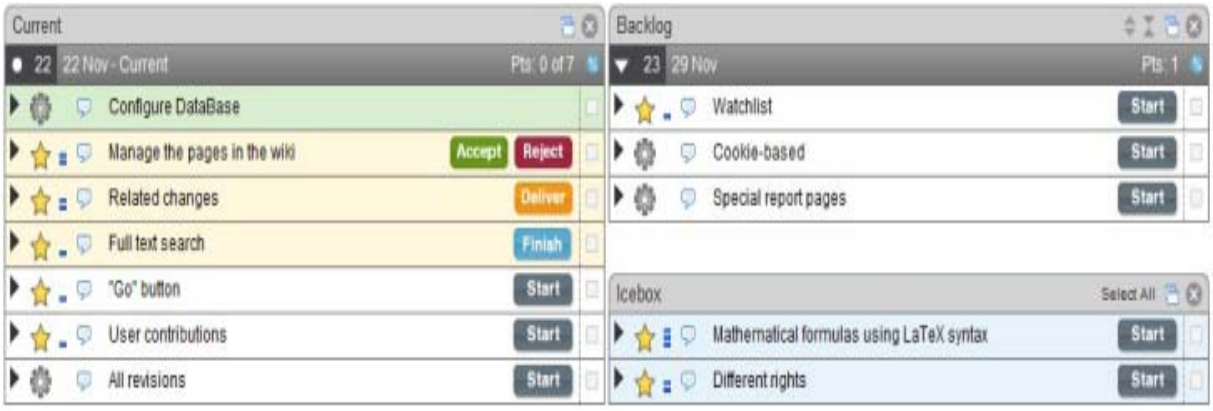

Figure 10. Example of an agile tool managing current activities, the waiting activities and possible future activities

The user may or may not request changes. If he requests it, the code written as shown in this study it may be easier to change, due to the acceptance/functional tests that will be always in execution to show where a requested change (or a new feature) crashes the application. Both the changes and new features are treated in a new cycle of the project, returning to the Requirements phase. If there are no changes or new features, the process cycle returns to the Planning phase, where the development team will have a new meeting with the PO to select the User Stories for the new iteration. Another activity to be performed at this phase involves a meeting of the development team to analyze how the Execution phase was and solve potential problems for the next iteration.

4. Conclusion

The agile process for Web development proposed in this study is a way of involving the disciplines of Requirements; Analysis and Design (Modeling); and Implementation and Testing (Construction), through the artifacts that are highly related. Another important contribution is that the study shows effectiveness in ensuring that the requirement requested by the user was actually implemented because of the use of the User Stories. Thus, the user may have a better understanding of the process and provide more support during the project.

The larger the system to be developed, the greater the difficulties in dealing with the high number of Story Cards and create the Model Navigation and then use them to transcribe the User Stories since this procedure requires a good visual aid. Thus, the process is suitable for Web applications to small and medium-sized businesses. Non-functional quality attributes (such as security and performance) are difficult to put in a User Story, also limiting the use of this agile process.

Future work includes improving the proposal and a more formal approach, as well as the study of a more effective way of dealing with non-functional quality attributes. Also include the construction of tools and other resources to support the proposed process. A case study will be designed to collect data on effort, time and maintainability degree in comparison to other agile processes.

References

[1] O'Reilly, T (2005). What is web 2.0: Design patterns and business models for the next generation of software. *O'Reilly Media*. [Online]. Available: http://oreilly.com/web2/archive/whatis-web-20.html.

[2] Schwaber, K (2004). *Agile Project Management with Scrum*, 1st ed. Microsoft Press.

[3] Beck, K., Andres, C (2004). *Extreme Programming Explained: Embrace Change*, 2nd ed. Addison-Wesley.

[4] Norrie, M. C (2008). Pim meets web 2.0. In *27th International Conference on Conceptual Modeling*, vol. 5231. Barcelona, Spain: Springer-Verlag, LNCS, p. 15–25.

[5] Cohn, M (2004). *User Stories Applied: For Agile Software Development*, 1st ed. Addison-Wesley Professional.

[6] Ceri S., Fraternali P., Bongio, A (2000). Web modeling language (webml): A modeling language for designing web sites. *In: 9th International World Wide Web Conference*, Amsterdam, Netherlands.

[7] North, D (2006). *Introducing Behavior Driven Development*, 1st ed. Better Software.

[8] Beck, K (2003). *Test-Driven Development by Example*, 1st ed. Addison-Wesley.

[9] Ross, D. T (1977). Structured Analysis: A Language for Communicating Ideas. *IEEE Transactions on Software Engineering 3(1)*, Special Issue on Requirements Analysis, p. 16–34.

[10] D'Souza, D. F., Wills, A.C (1998). *Objects, Components, and Frameworks with UML: The Catalysis(SM) Approach*, 1st ed. Addison-Wesley Professional.

[11] Anderson, D (2010). *KanBan*, 1st ed. Blue Hole Press.

[12] Kilov, H (1998). *Business Specifications: The Key to Successful Software Engineering*, 1st ed. Prentice Hall.

[13] Loyd, D., Rimov, M (2004). Expresso developers guide, *JCorporate Ltd*. [Online]. Available: http://www.jcorporate.com/expresso/doc/edg/edg.pdf.

[14] Pressman, R., Lowe, D (2008). *Web Engineering: A Practitioner's Approach*, 1st ed. The McGraw-Hill Companies, Inc.

[15] Martin, R (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*, 1st ed. Prentice Hall PTR.