



Agile Software Development of Embedded Systems

Version : 1.5
Date : 2006.06.08
Pages : 34

Authors

Marco Alessi
Giovanni Aiello
Alberto Sillitti
Giancarlo Succi

Status

Final

Confidentiality

Public

Agile Deliverable D.5.2.7

A new development process for Engisud

Abstract

After a careful analysis of the development process in Engisud, in this document XPAME, an XP-based Agile methodology for Engisud, is proposed. XPAME has been thought to meet the several critical issues gathered from interviews of the main representatives of functional, technical and development areas of Engisud.



ITEA

INFORMATION TECHNOLOGY
FOR EUROPEAN ADVANCEMENT

Index

Abstract.....	3
Chapter 1.	5
Agile methodologies for software development, new approaches for Engisud.....	5
1.1. Software requirements Management	6
1.2. Adaptive development process management - Iterations	7
1.3. Difficulties to make measures.....	7
1.4. Limitations to adopt agile methodologies.....	8
1.5. Some best practices of the proposed agile methodology for Engisud	8
1.5.1. Test Driven Development and Unit test practice.....	8
1.5.2. Refactoring.....	11
Chapter 2.	12
The current development process of Engisud	12
2.1. Interaction with the customer and requirements elicitation.....	12
2.2. Requirements analysis	14
2.3. Processes development: technical and development areas	15
2.4. Considerations.....	16
Chapter 3.	20
Proposed Methodology for Engisud.....	20
3.1. Proposed Methodology: XPAME practices.....	20
3.1.1. Basic concepts of Agile Requirements Management	20
3.1.2. User stories.....	20
3.1.3. Writing user stories	23
3.1.4. Estimation of user stories.....	23
3.1.5. Iteration and Release Planning.....	24
3.1.6. Acceptance tests.....	26
3.1.7. Relationship with the customer.....	27
3.1.8. XPAME development process	28
3.1.9. Tools to support XPAME.	32
Conclusions and future work.....	33



Abstract

In this document XPAME, an innovative software development process for Engisud, is proposed. XPAME supports an Agile project management and development and is based on some practices of the eXtreme Programming:

- User stories
- Acceptance test
- Test driven development
- Continuous feedback from the customer
- Team monitoring through the processing of several Agile metrics such as team velocity, iteration length and graphs showing the differences between the real team progress and the planned one.

The development process is supported by two tools for Agile software management, described in [4] and [5].

The above mentioned tools support a complete development process: project definition, user story definition, acceptance test definition, software development by Test Driven Development. The two tools devoted to the agile project management and development are called, respectively, *eXtreme Project Manager* [4] and *Test case generator* [5] and will be part of a framework available for Engisud at the end of the ITEA-AGILE project, expected on 31st December 2006.

Extreme Project Manager supports a complete agile Project Management through customer user stories definition and estimation, acceptance test definition, iteration plans, release plans, responsibilities assignment.

Test case generator supports a software development by Test Driven Development through the integration of one or more plug-ins to the development environment used in Engisud (Eclipse 3.0 or later).

In this document the terms “customer” and “team of customer” are used as synonyms.

During the agile assessment period a set of practices and assumptions, described to the *manifesto for agile software development* [6], were taken into consideration.

XPAME is an iterative and incremental process based on a set of agile best practices used on an XP-based process. The major practices are:

- To support a non rigorous requirement elicitation through an involvement of the customer (or customer team) to write *User stories* and *Acceptance tests* rather than to introduce use cases. The goal is to minimize the effort spent to write useless analysis and design documentation and to make it consistent with the code, in fact the inconsistent tie between them makes the documentation unusable. XPAME emphasizes the advantages of a requirements elicitation and analysis by user stories "directly written by the customer", acceptance tests and development of the code that satisfies the tests.
- To perform an *Agile Modeling* avoiding an up-front architecture definition, performing minimal sketches of the model and starting the development by Test Driven Development practice. Software tests (xUnit tests) are the reflection of customer acceptance tests and their satisfaction assures that the code meets customer expectations.
- To write test classes before the implementation of any business logic unit satisfying tests.
- To perform integration and regression testing.



- To develop high-level features of the software and, then, to evolve software components adding the needed flexibility and removing the needless one.
- To adopt the Pair Programming practice within the team.
- To evolve the software development towards the direction performing the maximum value added for the customer.

A first version of Extreme Project Manager and Test Case Generator has been released in March 2006. They follow all aforesaid principles, except the pair programming practice which is made at discretion of the project managers.



Chapter 1.

Agile methodologies for software development, new approaches for Engisud

The software development is often a chaotic activity characterized by continuous code updates due to continuous changes of the software requirements.

Furthermore, when several design choices are taken, a change of requirements during the development could involve a change of the design. These changes can introduce a low risk level when the software size is small, but when the size increases, it becomes hard to add new features to the software. Often a long test phase after the code development finds several bugs that have to be fixed, involving delays for releases and the customer dissatisfaction, due to the bugs control unpredictability.

Conventional software engineering methodologies impose a disciplined development process with the aim to make the process as efficient as possible setting great emphasis to the planning and the modelling. Nevertheless, these methodologies are affected by several drawbacks, due to their bureaucratic nature; too many tasks have to be performed to the software development, often involving a speed decrease.

Recently, a new group of methodologies has been proposed. For a brief period they have been known as *light-weight* methodologies, but now the new term is *agile methodologies*. They were an attraction for both software and industrial companies due to their agile responsiveness to bureaucratic features of plan based methodologies. Agile methodologies are *adaptive* rather than *predictable*, in order to obtain an agile responsiveness to requirement changes.

As a result, agile methodologies add the *non-predictable* feature and avoid redundancies on the code to conventional software development processes (i.e. *waterfall*) or, generally, to any plan-based methodology.

The major feature showed by agile methodologies is the incremental software development without any up-front model, and the evolvement of the model releasing software increments that imply value added for the customer. Agile methodologies are *code oriented*, since they assume as major documentation the implemented code.

The iterative feature of agile methodologies provides short-term plans inside the temporal window of a single iteration. Furthermore, agile methodologies are directed to the individual rather than to the process. For this reason they are *individual-oriented* rather than *process-oriented*, keeping in mind the developer's features, represented by his skill. For this reason a skill inventory compilation by developers adds efficiency to an agile methodology.

If the software development process has to be adaptive, high trust in the developers and their involvement in the decision making process is needed.

1.1. Software requirements Management

Conventional methodologies assume that requirements are well formalized before beginning the development process. In this phase a set of documents dealing requirements elicitation and analysis are produced (i.e. the Requirements Analysis Document). When all requirements are gathered and analyzed, the code development will be based on the analysis results, providing a design of the whole software. A plan-based approach can be reliable in a stable environment, where the software requirements are well known.

Figure 1 shows the cost of software requirements changes during the phases of a plan-based development process according to the Kent Beck analysis [7].

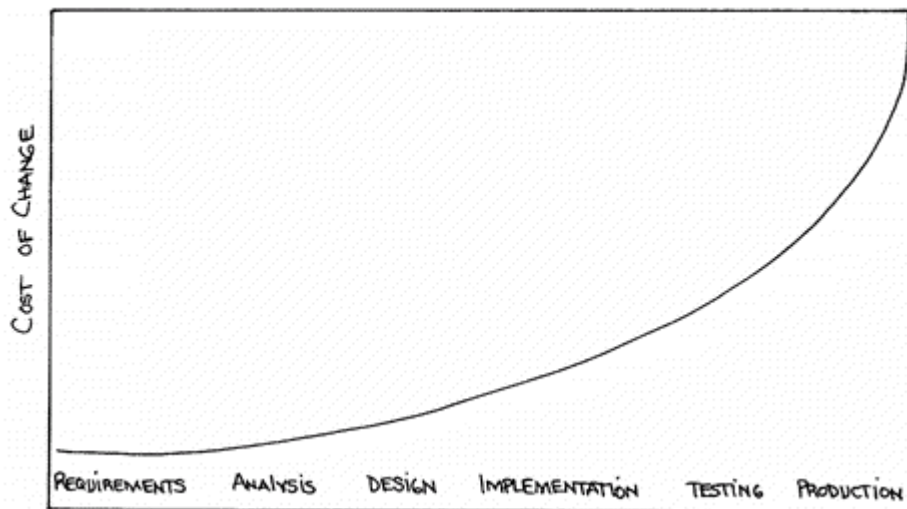


Figure 1. Cost of requirement changes on a plan-based process

In figure 1 it is clearly showed the exponential nature of the cost curve in a *waterfall* development process. Nevertheless, most of software projects are characterized by continuous requirements changes, making unreliable the predictable solution.

The main goal, therefore, was to make *adaptive* the Engisud software development process aiming to obtain a logarithmic trend of the cost of change during iterations (figure 2).

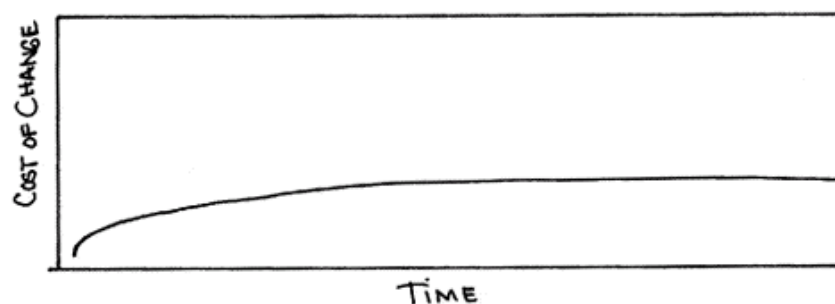


Figure 2. Cost of requirement changes on an agile process (main goal for Engisud)

Responsiveness to the requirement changes also involves a reduction of the time-to-market.

A careful analysis of the current development process of Engisud has pointed out the natural change of requirements during any business software realization, therefore the adoption of an *adaptive* development process rather than a predictable development process is needed. Furthermore, the



agile assessment has pointed out the need to continually release software increments to the customer. In this way the customer can use the software after the first iterations in order to evaluate it, providing continuous feedbacks. For this reason an iterative development process was proposed as an agile methodology for Engisud

The pilot project subjected to an agile assessment was the ERP-Light project because it involved the complete Engisud development process.

ERP-Light has been affected by continuous requirements changes involving a total system redesign. The eXtreme Project Manager tool has been realized to meet the business needs, to drive to an XPAME development process, to add the *adaptability* feature, to continually release software increments, to obtain continuous feedback from the customer in order to develop code satisfying customer expectations.

1.2. Adaptive development process management - Iterations

The agile assessment in Engisud has pointed out the need of a reliable feedback method among customers, management and development teams frequently showing the trend of project results. For this reason, XPAME is also an iterative process.

Each iteration is devoted to the development of a subset of functional requirements, according to descriptions related to user stories to be developed for the selected iteration. Therefore, the iterations provide increments to the existent code from the customer point of view. When a set of iteration results can form a release, the customer can start to interact with the software release, providing continuous feedback to the developer team.

The iterative development can be applied also in predictable processes contexts, but it is required on adaptive ones whose peculiarity is to respond to functional requirements changes with a minimum cost.

The iteration length depends on the chosen agile development process: one to three weeks on eXtreme Programming, one month on SCRUM methodology, and more time for Crystal family methodologies. However, the goal is to make iteration as short as possible in order to provide continuous releases and to obtain continuous feedback from the customer.

1.3. Difficulties to make measures

Often, project managers do not implement code, they rather assign responsibilities to developers. Therefore a reliable way to monitor the software progress is needed in order to check possible differences between the planned progress and the real one.

Nevertheless, performing these measures is a difficult task due to the non-predictable team features. Although the spent effort, often to perform basic software measures, such as the productivity, is very difficult, without which any external control type is destined to fail.

The introduction of a measures-based management without good measures implies several problems. Furthermore, a measure of performances requires the evaluation of all considerable factors. The omission of some details involves the alteration of the work from workers in order to gather best measures, even if the work effectiveness decreases.

Therefore, there is to choose between a measure-based management and a delegation-based management. In a delegation-based management workers decide how to perform their work.

The measure-based management is right for simple and repetitive works, with a low knowledge level and with measurable results, just the opposite of the experience of the software engineering.

Therefore, traditional methods assume that the effective management is the measure-based one.



Nevertheless, the agile community assume that a measure-based management involves an alteration of measures themselves, while a delegation based management is more effective.

All these aspects are supported by the proposed system.

1.4. Limitations to adopt agile methodologies

Agile methodologies are thought out for small/medium size development teams.

The eXtreme Programming methodology expects team composed by approximately twenty people.

Other agile approaches are thought out for more large teams, for example the Feature Driven Development was primarily designed for teams composed by approximately fifty people.

Even if the software project is not characterized by requirements changes, several agile best practices would be adopted on the current development process in order to reach a risk decrease and a fast development process. The Test Driven Development best practice assures to implement working and tested code. Furthermore, a set of test types (black box testing, white box testing, integration testing e regression testing) assures the full code coverage by a testing of all software components.

1.5. Some best practices of the proposed agile methodology for Engisud

1.5.1. Test Driven Development and Unit test practice

The literature concerning software development is rich of pattern and best practices; but not always what is formalized in literature finds a correct and systematic application in the development practices. To the practical act, important themes are often not included in a systematic way inside the development process. One of these themes, maybe the most important, is related to the tests. The test is verification, a validation of the work in terms of correct development of the functionality of the system from the customer's point of view.

Several testing typologies exist:

- *White-box testing*: also called building test that aims to verify if the class is well built giving incoming varied types of data and testing which exceptions are not thrown.
- *Black-box testing*: also called functional testing, used to verify that the software corresponds to its specifications and that all the functionalities expected are included.
- *Regression testing*: useful to verify that all defined tests pass whenever a code update is done. Therefore, test can be ran when a new feature is added or during a code refactoring, in order to assure the well working of the software and the lack of new bugs.
- *Integration testing*: integration tests are useful to verify that all software components correctly communicate among them. For instance, tests concerning a control class are integration tests.

Agile methodologies underline the role of the testing in the *Test Driven Development* (TDD) practice. Test Driven Development is a technique which tends to express software functionalities in terms of tests.

The tests are written before the business code implementation and they should contain the verifications to establish that the code behaviour fits with customer requests.



TDD features are:

- To provide Unit Tests which correspond to software specifications.
- Every test has to verify a functional code unit.
- To create a test suite that describes requirements and that reflects acceptance tests written by the customer. These tests will initially fail because specifications have not been implemented yet.
- To write the code which satisfies specifications aiming to satisfy tests.
- To perform a “Refactoring” to assure good code implementation in terms of both a lack of redundancies and its read easiness. Refactoring is very important because it allows to take design choices and to implement particular design patterns.

To perform a development based on a translation of customer requirements in software tests produces several benefits in terms of:

- *Quality*: TDD assures to implement code with minimum bug number related to functionalities to be realized, but a set of *well written* unit tests has to be available. Therefore, the unit test should be driven by the technical area, which holds greater experience in development and software design.
- *Functional Requirements Documentation*: Each unit test describes appropriate use of the single developed class.
- *Maintenance*: To have well written tests means to continuously maintain the quality of the code without the risk of compromising already working code, involving low maintenance costs.
- *Continuous feed-back*: A cycle in TDD should be very short so that the feedback level is high.

The TDD work-flow is the following:

1. To perform a minimal sketching of functionalities to be realized;
2. To implement interfaces of classes to be developed;
3. To add a test before the functionalities coding;
4. To run the test (the first time it will fail);
5. To implement the code related to each functionality;
6. To run the test again;
7. If the test fails, to modify the code implemented on step 5 and to run the test again (Step 6)

8. If the test passes

8.1. To verify, through refactoring, the lack of redundanciess and needless statements and to run the test again (Step 6).

Go back to step 1 for the development of a new functionality.

Figure 3 shows the TDD process.

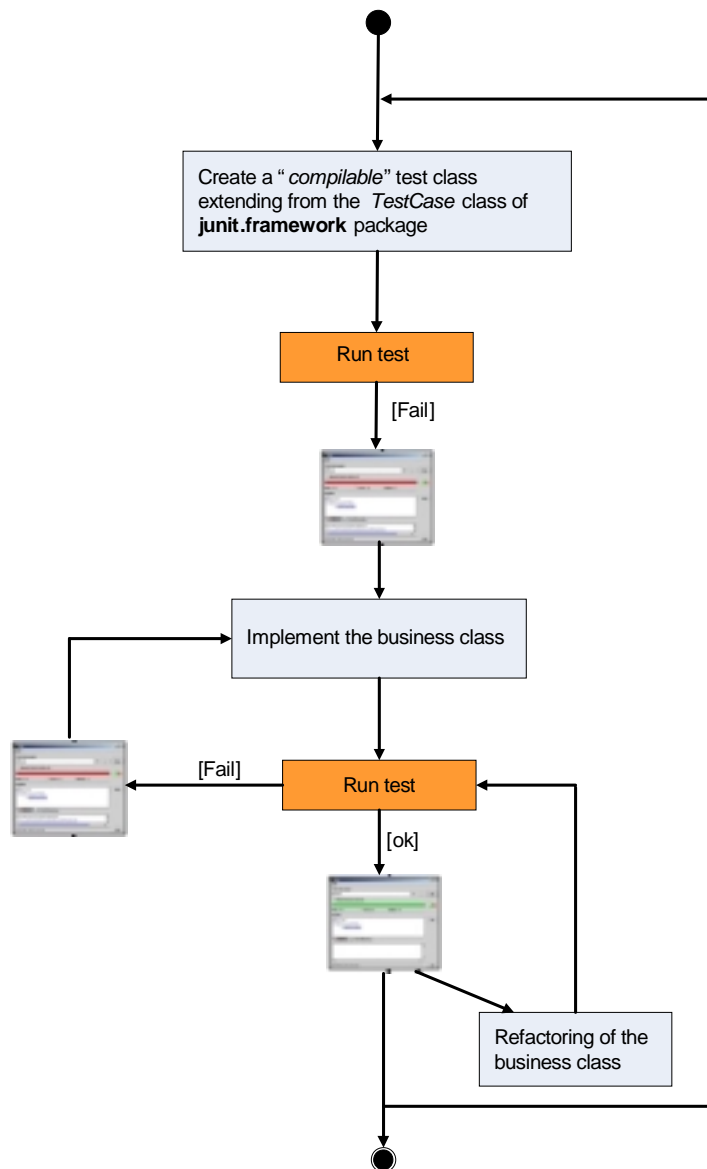


Figure 3. Test Driven Development

Several frameworks such as JUnit [1] are very useful in order to make the TDD practice and tests execution and management easy.

In TDD, the index of test quality is "directly proportional to the amount of bugs found", therefore if initially the tests do not point out any bugs, it means they are not sufficiently rigorous.



Furthermore, it is a good practise to add tests related to bugs found before removing them. Even if agile methodologies are not used in the development process, to develop software according to TDD guarantees the availability of a test suite associated with the implemented code must exist, increasing the robustness, simplifying the refactoring phase and the introduction of new functionalities through continuous integration. TDD, opportunely applied, guarantees a test suite allowing to check software regressions, and provides documentations of the implemented code, in fact the tests are, of course, good “*use cases*” of implemented classes and methods.

1.5.2. Refactoring

A code refactoring means to perform several changes to the code keeping characteristics unchanged, so that an improvement of non functional features such as simplicity, flexibility, comprehensibility, performances and the robustness of the code is performed.

Therefore, refactoring represents a disciplined way to increase the elegance of the existing code with a lack of new bugs, it adds comprehensibility of the code so that a better management of the code in terms of maintenance, bug fixing and addition of new functionalities is realized.

The refactoring consists of several steps of change of the code, whose cumulative effect can drastically improve the entire software project. Every step has to be simple and clear (i.e. to move a method from a class to another class). A pre-requisite of the refactoring practice is the availability of a test suite able to check software regressions.

The tests are the principal tool that allows to modify working code without any risk.



Chapter 2.

The current development process of Engisud

In this chapter an analysis of the current development process of Engisud is described. The analysis involves all phases of the current process: interaction with the customer, requirements elicitation and analysis, software development, software testing. The piloting refers to the ERP-Light project. During the analysis several issues were identified and an agile development process was proposed for Engisud in order to add some agile best practices to its current development process. Since the agile development process uses several XP best practices, in this document it is called XPAME.

2.1. Interaction with the customer and requirements elicitation

During the requirements elicitation, Engisud performs a set of interviews in order to identify the business process and area of the customer, usually represented by Small/Medium Enterprise (SME). Therefore Engisud identifies and formalizes the SME business process, the ICT system, and critical issues.

Such activities are difficult tasks because the customer is not always able to explain its needs, therefore Engisud also aims to drive the customer to express real problems and priorities. All phases related to the interaction between Engisud and the customer are shown in figure 4.

The above mentioned activities concern the *Focus* phase.

The next phase is the *As is*, where a rigorous analysis of current internal processes of the customer company is performed (i.e. marketing, warehouse management, purchases, manufacturing, accounting area). Engisud proposes its solution based on the results of the analysis, and a further verification of critical issues is performed in order to find problems not underlined by the customer. After these activities the customer provides the so-called “acceptance scenarios”, and at this point the involvement of the customer decreases.

The interaction with the customer begins again when the *To be* phase starts. In this phase Engisud shows the customer the proposed system through “use cases” that perform processes, and the value added is underlined.

This activity is made easy thanks to the high level of know-how present in Engisud related to processes-based development that is a well thought by potential customers.

The output of the *To be* phase is a document containing the Engisud solution. This document describes improvements to be carried out on the current informative system in terms of processes; furthermore it describes how Engisud will work in order to resolve problems that the customer did not identify.

Finally, the *Plan* phase is devoted to the design of a new process scenario, to perform several estimates in terms of effort and costs to realize each scenario. All plans are discussed with the customer.

The Plan phase is also devoted to identify “facilitated finance solutions” in order to make the purchasing of Engisud solutions less expensive for the customer.

All the above mentioned phases are applied in an iterative way, based on a priority order provided by the customer. Thanks to this feature, executable software increments are frequently released to the customer and can be evaluated.

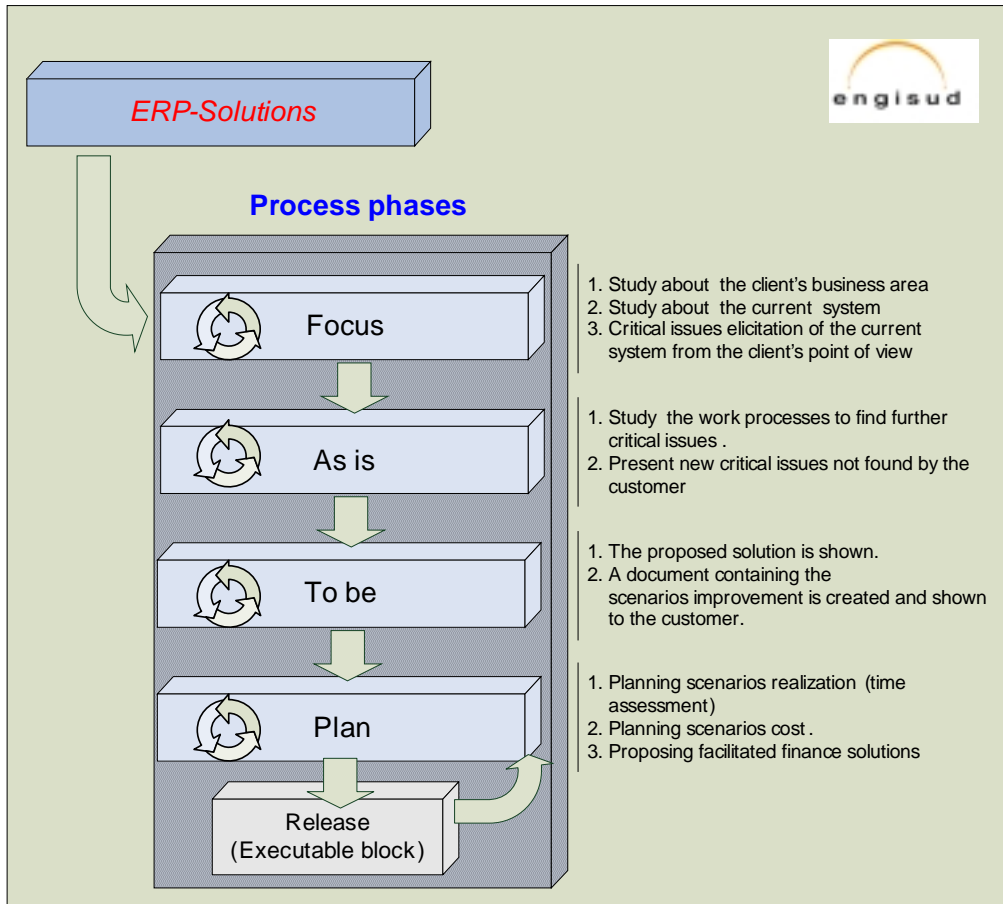


Figure 4. Phases related to the interaction between Engisud and the customer

Engisud proposes both ERP solutions and Oracle solutions. For the latter the *Electronic Business Suite Special Edition* version is proposed.

When Oracle solutions are proposed, Engisud realizes customizations, for instance it performs custom user interfaces for the customer company.

In order to identify additional features, during first meetings with the customer a set of questionnaires are submitted so that the customer can fill them in with the support of the functional area of Engisud.

Furthermore, Engisud performs the Oracle suite installation and functional tests.

After the installation, the customer provides several “acceptance use case” so that Engisud can test the code from the customer’s point of view.

Nevertheless, the customer does not provide any acceptance tests because the proposed product has standard features.

A critical issue during the requirements elicitation is the lack of the participation of the customer.

2.2. Requirements analysis

Concerning the requirements analysis, Engisud adopts an incremental method (similar to Boehm spiral model).

The first experience for Engisud related to ERP solutions is represented by the ERP-Light project.

The ERP-Light project, because of its largeness, has been subdivided into several functional areas each of which has been assigned to a single technical designer. For each functional area *a processes based design* has been performed.

Each process provides the following output types:

1. Web Workflow (WWF)
2. Process Workflow (PWF)
3. Software Component (CSW)

Note: WWF, PWF and CSW follow a standard

WWFs represent process units, therefore these “*work units*” are tested through unit tests.

The PWF represents the description of the product, it contains a set of all WWFs and CSWs involved in the process.

For each process two documents are provided containing, respectively, the process description and the “*unit test plan*” which a single WWF has to pass.

The two documents follow a life cycle, described in [2], before their publication to the technical area.

When documents are published, the technical designer begins the WWF development.

The requirements analysis is shown in figure 5.

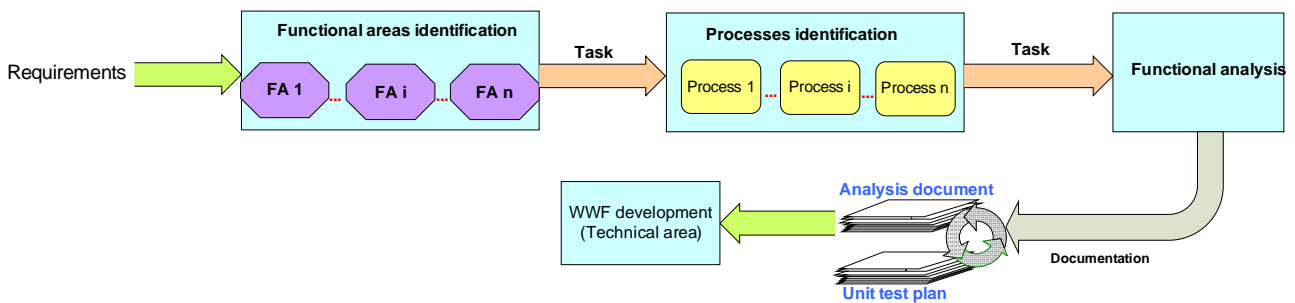


Figure 5. Requirements analysis process (functional area)

During interviews the need to have more feedback from the customer, in order to provide a “better team work integration and to speed up the analysis process”, was underlined .

2.3. Processes development: technical and development areas

The technical area is made up of technical designers which can access the documentation related to several Web Workflows provided by the functional area.

Technical designers examine documents provided by the functional area and they assign *activities* to the development area. Each activity is assigned to a single developer.

At the current stage, no testing tools (i.e. JUnit) are used to perform unit tests of a single activity; when the development area implements a single activity, it performs functional tests described in the unit test plan document provided by the functional area.

When unit test plans are passed, the development area assigns the activity to the technical area which performs both the integration of the activity on the system and the integration testing. In the latest phase, the JUnit framework is used.

When all tests pass, the technical area assigns integrated WWFs to the functional area which performs all tests again.

The development process is shown in figure 6.

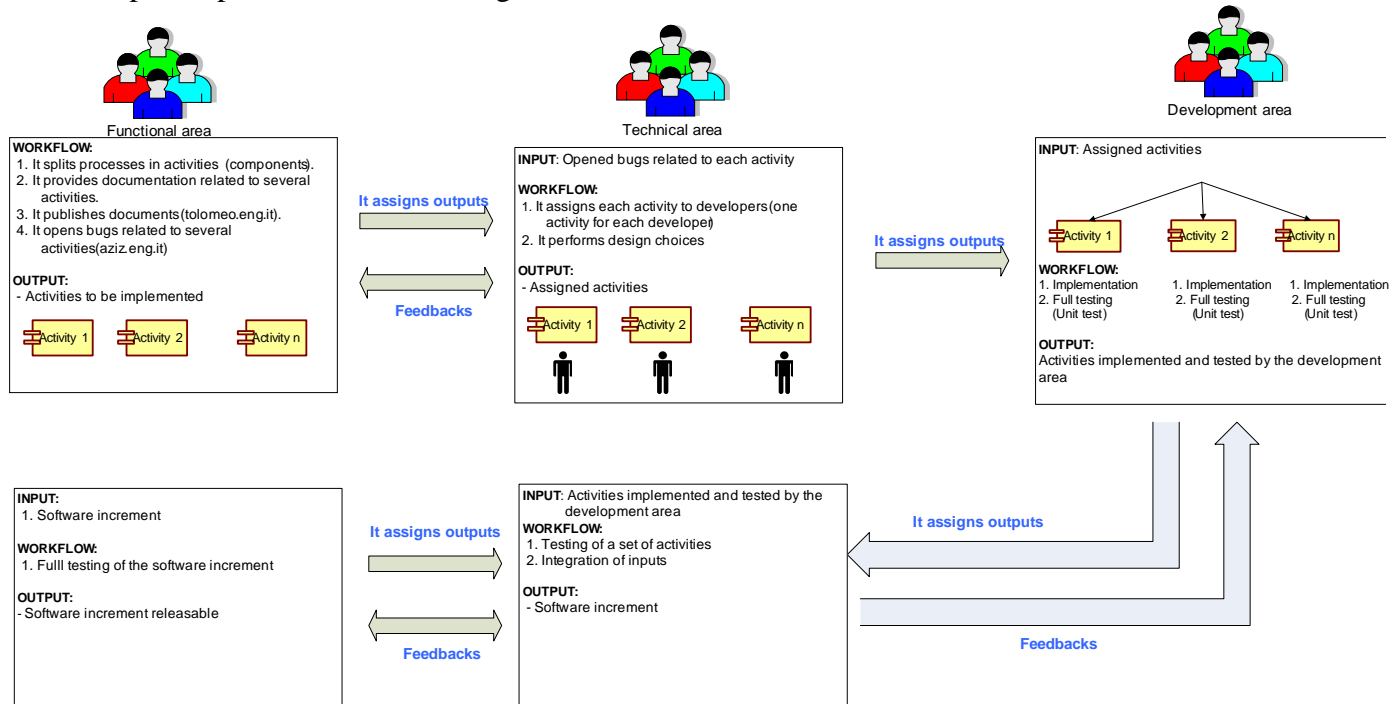


Figure 6. Organizational diagram related to the execution of several activities

Every 15 days, a meeting among functional, technical and development areas is performed in order to provide a Work Progress State (WPS). During the WPS meeting there is a debate regarding developed activities and an identification of problems related to delays and/or technologies. WPS is also a good way to perform a planning for the next 15 days, nevertheless requirements changes may involve the plans reliability.

Assignment criteria of activities depend on their overhead and on the developers' skills. This well meets with the agile methodologies principles.

Currently, Engisud has a small/medium size development team (about 10 people). This last feature facilitates the deployment of agile best practices.



The technical/development areas do not adopt an automatic testing method. Apart from this consideration, technical and development areas do not present critical issues.

2.4. Considerations

The analysis of the entire development process has identified some critical aspects inside a few macro-phases: requirements collection, requirements analysis, development and testing.

The requirements collection phase, that at present deals with Oracle solutions but it can be considered a general process, highlights some critical aspects related to the direct interaction with the customer. At present, his involvement is low and concentrated only in the requirements collection and in the testing phases. The customer does not provide any feedback during the development and he does not provide any acceptance test during the requirements collection. This approach results in difficulties in verifying the developed modules and the customized solutions. Acceptance test would be an opportunity to identify and ship only the functionalities that provides value to the customer reducing the overhead associated to whatever seems to be necessary but is actually not required.

During the analysis of the development process, we have asked the heads of all the development areas of Engisud to fill in a set of questionnaires. Such questionnaires have also been submitted to other 64 software companies located in the European Union and in North America. From the analysis of the answers collected from Engisud and from the other companies, it is possible to identify problems inside Engisud and positioning Engisud in a wider context. This positioning is useful to understand if such problems are only related to Engisud or they are common ones.

	Engisud	Other 64 companies
Have you modified the process or the technique used to collect requirements in the last few years?	Yes, because of the availability of new and better techniques	100% yes. The main reasons are supporting changing requirements and new and better methods for their management
Is the requirements collection performed mainly at the beginning?	Yes	No, during the entire project (100% agile companies and 10% traditional companies). Yes, 90% traditional companies.
Techniques for requirements collection	Well known techniques such as questionnaires, interviews, documents analysis and group techniques such as workshops.	All companies use well known techniques such as questionnaires, interviews, etc. 100% of the agile companies and 50% of the traditional ones use also techniques based on simulations or models
Do you use automated tools to manage requirements?	A few tools for data management are used to support the development of functional tests, trace requirements and support requirements analysis.	50% of the traditional companies and 60% of the agile ones use automated tools to trace requirements.
Do the requirements	Yes., sometimes the	For 60% of traditional companies and

change during the development?	requirements identified at the beginning change due to the increased level of understanding of the problem and the identification of errors and/or misunderstandings	100% of the agile ones, requirements change frequently or very frequently. The main causes are: lack of clarity in the objectives (>80%), communication problems (misunderstandings) (>60%), improvement of the level of knowledge of the problem (60%)
What is the impact of change?	The main impact is on the organization and control of the development process, in contracts (time and cost), and customer satisfaction	For 60% of the traditional companies it has an impact on contracts, for 50% on the quality, for 45% on the customer satisfaction, for 30% on the organization of the development process. For 100% of the agile ones the impact is not relevant
Are there communication problems? How are they solved?	Yes, with the customer due to the different languages used and due to the communication channel. Problems are solved trying to use a common language, asking more detailed questions, using diagrams and workflows.	About 45% of the companies have problems in the communication with the customer due to the language or due to the lack of domain knowledge
Are there any conflicts?	Yes, between analysts and developers caused by the different points of view, differences of the people involved, emotional and relational aspects. Conflicts are solved through frequent meetings and identifying the reasons for them	45% of the traditional companies and 60% of the agile ones experience conflicts between developers and managers, 10% of the traditional and no agile companies experience conflicts among developers, 50% of the traditional and 20% of the agile companies experience conflicts among the representatives of the functional areas. For about 60% of the agile companies such conflicts are related to the traditional culture of the companies. For 60% of the traditional companies the conflicts are due to the lack of information or due to the differences among the members of the development team
What kind of collaboration do you have with your customers?	The customer is not present during the development process due to his lack of time	65% of the agile companies and 80% of the traditional ones have a continuous relationship with the customers, 35% of the agile ones and 20% of the traditional ones have the customer on-site.

The data analysis identifies the following problems:

- Misunderstandings in the collection of the requirements.



- Instability of the requirements collected at the beginning.
- Lack of verification of the requirements by the customer.
- Inability of the customer to provide a clear and complete set of requirements actually needed.
- Difficult relationship with the customer due to the differences of the involved parties (knowledge, language, etc.) and due to the lack of time of the customer. This last problem is usually solved (partially) through meetings on the most important issues.
- Sometimes the identification and the management of the requirements create conflicts among the subjects involved in the project.
- Conflicts related to the analyzed requirements due to the absence of the customer causing lack of information.

Comparing Engisud with the other companies analyzed, Engisud has the same problems of the traditional companies. There are no relevant differences in all the questions.

About the software development process, the common problems are listed in the following table.

	Engisud	Other 64 companies
Have you changed your development process in the last few years?	Yes, because of the change of the technologies used	100% of the companies have changed it. The main reasons are: changes of the technologies used, changes in the customer needs
What are the main development problems?	1) Difficulty in shipping the product according to the plan 2) Development of all the requested features	1) Difficulty of shipping the product in time with all the requested features (70%) 2) Relationship with the customer (35%) 3) Lack of personnel (>25%)
Which are the adopted solutions?	Incremental releases and more involvement of the customer in all the development phases	Traditional companies: more detailed requirements analysis (50%), incremental releases (70%), more communication (20%), Agile companies: more agile methods (70%), more involvement of the customer (40%).
How is organized the development process?	Precise division of the development phases and incremental development based on the feedback of the customer	Traditional companies: incremental development with feedback from the customer (55%), precise division of the tasks (45%), prototypes (45%). Agile companies: incremental development with feedback from the customer (75%), planning of only needed functionalities (45%)
Are you satisfied with your planning abilities?	Not much.	Very much: 15% agile, 10% traditional Much: 75% agile, 65% traditional Not much: 10% agile, 20% traditional Not at all: 5% traditional
How do you plan?	On the base of the past	Traditional companies: past experience



	experience	(80%), function points (15%) Agile companies: past experience (70%), function points (30%)
--	------------	--

Comparing Engisud with the other analyzed companies, Engisud shows the same problems of most of the traditional companies. However, in this case, there are some important differences. The main ones are:

- Engisud has no problems in recruiting qualified people. This is because it is located in a geographical area with a large university and young graduated have difficulties in finding a good job as computer scientists in the area.
- The level of satisfaction with planning is poor. This is mainly due to the specific customers of Engisud: SMEs. Often, SMEs have problems in identifying their needs or they change their expectations during the development. This implies a frequent re-planning of the project after the requirements collection and analysis, therefore there is a need for development methods able to support changes since it is not possible to avoid them.

Chapter 3.

Proposed Methodology for Engisud

On the base of the analysis of the questionnaires, this section proposes a methodology based on some best practices of the agile methods and adapting them to the specific context of Engisud. To support such practices, a tool for project management is proposed. Such tool is able to support the agile practices adapted to Engisud. In this first version XP projects are supported.

The developed tool includes two modules: eXtreme Project Manager [4] and Test case Generator [5]. Such modules guide the developers through a process a customized version of XP.

In this section, the present development techniques and the proposed new one are presented. The *light-weight* features of the new method and the usage of the tools are highlighted.

The terminology and the main concepts of XP will be introduced contextualizing them in the proposed development method. Such concepts include, among the others:

- Requirements collection and analysis: user stories, acceptance tests.
- Software development and testing: Test Driven Development.

Among these key concepts, iterations and releases plan will be analyzed to guide the development team to improve their planning abilities and ship incremental releases to collect valuable feedback from the customer. Such approach is required to solve the problems related to the frequent changes of the requirements.

To support the development method, the usage of the eXtreme Project Manager will be analyzed.

3.1. Proposed Methodology: XPAME practices

3.1.1. Basic concepts of Agile Requirements Management

The basic concepts described here below include the assumptions made during the development of eXtreme Project Manager and their meaning in the specific context of Engisud.

3.1.2. User stories

A user story describes valuable functionalities from the point of view of the user of the final software product.

A user story includes three main aspects:

- They remind the assumptions made during the requirements collection.
- They are discussed with the customer to identify the details.
- They have some test connected to verify their correct implementation.

Since a user story is written down on a piece of paper, these three aspects are described as a Card, a Conversation, and a Confirm [3].

The Card is the row paper of the story, the Conversation is the content, and the Confirm is how to test the story.

User Stories are described by the customer not by the software analyst. This is because a story has to be written using an understandable business language rather than a technical language not well



understood by the customer. In this way the customer is forced to understand and express in a better way his own needs.

As described later, the customer decides the priority of the stories he has designed. Such prioritization will be the input for the planning of the iterations and the deadlines for the releases.

Here below there is a user story written for the development of the eXtreme Project Manager tool.

User story:

The developer defines the user stories in collaboration with the customer.

User Stories are functionalities that provide a real value to the customer. For this reason, stories containing both functional and non-functional requirement, as reported below, are not *well formed*:

- The product has to be written in Jana using SWT/JFace
- Use the MySQL Database

To include such information two approaches are suitable:

- a) Use additional stories.
- b) If some technical details are deeply connected to a story they can be added as notes, not as main content.

If a user story is particularly long it is defined as *epic*. The identification of epics is done when their length is estimated. In such cases they are split into smaller stories called *splits*. During the estimation of the stories the presence of the customer is important to make a correct split when needed to discuss the details.

An example of user story with a note is the following:

User story:

Information related to the release plan has to be printable

NOTE: The output has to include information related to the total number of story points, in which iteration is planned the shipment, and information related to the velocity of the team

The definition of the tests (used for the Test Driver Development) is related to the definition of the *Acceptance Tests* described in the user story by the customer.

Even the acceptance tests are written together with the customer.

This is an example:

Acceptance test

- Building a release plan without user stories (is not allowed)
- Building a release plan with not estimated stories (is not allowed the inclusion of not estimated stories)
- Building a release plan without the computation of the velocity of the team (consider a team with virtual developers and create estimates)
- ...

Acceptance tests should guide developers in the definition of the unit tests in the first phases of the test driven development.

When all the tests are executed correctly, the developer can consider the related story “*COMPLETED BUT NOT VERIFIED*”.

When the entire team (project manager + developers) will become confident with the agile development they will realize that using user stories is more productive than use cases for the following reasons:

- User stories focus on oral and concise communication rather than on written and long communication.
- User stories are understandable by both the customer and the developers.
- User stories allow planning of releases and iterations.
- User stories are good for iterative development.
- User stories postpone the specification of the details until requirements are clear to the development team.
- The related acceptance tests are the base for the development of the test cases.
- From the test cases is possible to use Test Driven Development.

Since user stories focus on oral communication rather than on written communication, important decisions are not hidden into long documents but they are visible in the acceptance tests that have to be executed to consider the story implemented correctly.

An iterative development process allows the refinement of a software product through several steps: the development team produces the first implementation with only a few functionalities that are considered important for the customer. Such first release will be extended several times adding a few functionalities each time according to the priorities specified by the customer.

User stories are suitable for iterative development because it is possible to iterate through stories: to define a functionality the customer can write an epic and when it has to be implemented it can be split into several splits for which it is possible to plan the implementation.



3.1.3. Writing user stories

User stories can be implemented in an iterative way and they can be modified by the customer if needed. For this reason, techniques for lightweight collection and management of the stories are needed. Some techniques are the following:

- Interviews
- Questionnaires
- Observations
- Meetings for writing user stories

The first three techniques deal with the collection of requirements through a proxy (usually a *business analyst*) as it happens in the traditional techniques such as waterfall.

The meeting for writing the stories is specific for the agile development and involves developers, users, and managers. This meeting is usually done before each release plan. During that the participants write the stories without assigning a priority.

3.1.4. Estimation of user stories

During the estimation of the user stories it may happen to identify epics or it is required to modify some of them. In the former case, it is possible to split the stories to allow a better estimation and reduce the effort needed for the first implementation. Such operations can be done in any moment during the project.

The unit used for measuring the length of a story is the *story point* which is a day of development of a senior developer. The day is considered made of 8 hours dedicated completely to development.

For every user story two estimates have to be made: one at 50% of confidence level and one at 90% of confidence level. Such information are very important to create a meaningful estimate of the story [3]. The two measures indicate that the development team forecasts that the implementation will be done in that time interval.

Each estimation method for user stories is based on different sets of metrics for measuring the velocity and the effort of the team members:

- **Team velocity:** it is the number of story points that the team is able to complete in a working day. To calculate the team velocity developers have to provide estimates of the number of story points that they can implement in a day.
- **Iteration length:** is the number of working days of the iteration. Usually, it is about 20.

There are several techniques to estimate user stories:

- *Gut feel estimate:* is a qualitative estimate without a definition of specific values
- *Analogy estimate:* the estimate is based on similar user stories already implemented are considered. However, this estimate depends on the specific team.
- *Decomposition estimate:* sometimes it is useful to decompose a user story into smaller ones in order to make an estimate. At the beginning of the release plan user stories have to be small enough to allow reliable estimates and do not require too much time for their implementation. To allow a high level of flexibility user stories have to be quite small and their implementation have to be feasible in one iteration. If the estimate of a user story is greater than the velocity of the team in the iteration, the story has to be split to allow its

implementation is a single iteration. During the iteration plan, described in the next session, tasks for the user story can be defined and assigned to the team members.

- *Wideband Delphi estimate*: as described in [3], it is an iterative and collaborative process that a team uses to estimate the effort that will be spent to realize a set of customer user stories. Generally, actors involved in a Wideband-Delphi meeting are developers, project managers and customers. Few days before the meeting starts, project managers provide, to involved developers, user stories which will be estimated so that they can think about them. Therefore, estimators can be both developers and project managers. The participation of the customer is very useful because she/he can directly interact with developers and share opinions concerning user stories. In a Wideband-Delphi meeting a moderator (generally a project manager) manages the order of user stories to be estimated based on their coherence. When the meeting starts, each estimator read the user story and, in private, provides his 50% and 90% levels of confidence estimates in story points (iteration 1). When all estimators are ready, they publish their estimates. Initially, the estimates will be divergent, the estimators will therefore debate on the reasons of their estimates. It represents a good chance for knowledge sharing among several team members and to clear up misunderstandings, involving also the customer. After the group has discussed the story the moderator asks everyone to re-estimate, keeping old estimates (iteration 2). The estimators erase their old estimates and write new ones on their cards. When all estimators are ready, the estimates are republished. In many cases the estimates will already converge by the second round, otherwise high and low estimators explain the thinking behind their estimates. Iterations go on until estimate will converge, obtaining the final 50% and 90% levels of confidence estimates.

The process is repeated for the next user story until all user stories for the iteration are estimated.

3.1.5. Iteration and Release Planning

After the estimation phase, it is possible to define iterations and release plans. A release is made by one or more iterations.

The *Release Plan* is a way to define when functionalities are added to the product. Such functionalities are defined through a set of user stories and the related acceptance tests. The *Iteration Plan* is a way to specify which user stories will be implemented in a single iteration. Such plans are related to the team velocity and the size of the user stories.

Both the customer and the development team should be involved in the development of such plans. During the Iteration Planning, the customer orders the user stories according to the following criteria:

- The importance of the feature.
- The level of connection of a user story with the other ones. For instance, during the development of the eXtreme Project Manager tool, the user story “*Insert Task*”, describing the possibility of defining tasks, should have a comparable priority of the user story “*Insert Sub-Task*”, describing the insertion of sub-tasks.

The sorting of the user stories should be performed in collaboration with the developers to clarify technical consequences of the decisions of the customer.

It is not possible to define the release and the iteration plans if user stories are not already estimated.

The estimation of the stories is the assignment of story points to the story in order to define its size and the complexity. This session can be done without the customer since the estimation is only related to technical factors.

The eXtreme Project Manager tool supports the estimation of the user stories through *Wideband Delphi* sessions.

The Release Plan is created assigning a set of user stories to a release. On the base of the *iteration length* chosen by the Project Manager and on the base of the team velocity, the release plan provides information related to the time required for the implementation of the selected user stories specifying in which iteration the release will be ready and an estimate of the number of days required for the development.

The Iteration Plan is specific for a single iteration and it is the assignment of a set of user stories, ordered by the customer, to the developers considering the number of story points. On the base of the number of story points that the team can implement in a single iteration, the customer specifies the stories to implement.

For instance, let's consider the set of user stories related to the eXtreme Project Manager tool listed in Table 1:

ID	User story (title)	Estimate at 50%	Estimate at 90%	Partial sum (max-min) ²
1	Database Design	2	3	1
2	Definition of the user stories	2	3	1
3	User stories splitting	2	3	1
4	Release Planning	2	4	4
5	Effort estimation	2	3	1
	TOT	10	16	8

Table 1. Estimation in story points of a set of user stories

Supposing that the project manager is considering the *iteration length* equals to 5 days and the team velocity is 6 story points per iteration, the customer can assign a number of user stories equivalent to 6 story points per iteration.

To get an estimation of the time required for the implementation of the stories the *buffered schedule* is calculated as follows:

$$buffered\ schedule = \sum_{i=0}^n \min_i + \sqrt{\sum_{i=1}^n (partial\ sum)_i}$$

Therefore, if the customer chooses the first two stories:

$$buffered\ schedule = 4 + \sqrt{2} = 5.4\ story\ points < 6\ story\ points$$

The customer can add a *tiny story* to reach the maximum of 6 story points.

Now it is possible to define the release plan that lists the implementation time showed in Table 2.

Iteration	User story	Buffered schedule	Partial sum of story points
Iteration 1	1-2	5.4	5.4
Iteration 2	3-4	6.2	11.6
Iteration 3	5	3	14.5

Table 2. Release Plan

Looking at the table, the selected user stories will be implemented in the third iteration. The Buffered Schedule of the second iteration is 6.2 story points and the team velocity is 6 story points. Estimates are affected by error and a bias of ± 0.5 story point is acceptable.

If the team velocity is exceeded of more than $|\pm 0.5|$, it is possible to divide a user story into smaller ones in order to use the entire team speed for the iteration.

Team velocity is measured on the base of the estimates made by developers considering the effort they can spend in an iteration.

Determining the team velocity is a key preliminary step for every kind of planning (Iteration Planning or Release Planning).

In the release plan, if resources are not assigned, it is possible to make an estimate of the release plan assigning virtual developers to the project and making the estimated based on them. This approximate release plan can be useful to provide to the customer a general idea on the time required for the implementation.

After that, project managers assign real developers to the project in a way that the real team velocity is similar to the virtual one.

To plan a release, a set of stories are selected and ordered in a stack according to their priority. In this way knowing the team speed it is known at which iteration the project will end.

Before starting an iteration, the customer can make any modification to the plan and to the stories not yet implemented. Then the iteration is over it is possible to measure the real velocity of the team and monitoring how the project is evolving.

On the base of the real team velocity, the stacks of stories for the future iterations can be modified adding or removing stories according to story points that can be implemented in the iteration.

3.1.6. Acceptance tests

Acceptance testing is the process able to verify if the user stories have been implemented correctly. Acceptance tests are written by the customer in collaboration with the testers of Engisud. According to the organization of the company, the role of tester can be played by a project manager, a developer or anyone with the technical background. This is very important since he is the interface of the company with the customer.

The tester have to participate to every meeting with the customer since he is the person who interprets the expectations of the customer into JUnit test cases and guides the customer in the development of valid acceptance tests.

This procedure includes both the definition of acceptance tests and the execution of such tests with an automated tool for their verification such as JUnit.

Tests should be written just before the beginning of the iteration (or at least at the beginning) to assure that the development is carried out looking at the changing requirements.

Let's consider the following user story written for the eXtreme Project Manager tool:

User story:

It should be possible to define the iteration plan in a guided way

and the related acceptance tests:

Acceptance tests

- Try to start an iteration plan without defining iterations (not possible)
- Try to start an iteration plan without calculating the team velocity (not possible)
- Try to select stories of 7.0 story points with a team velocity of 7.2 (possible)
- Assure that every member of the development team receives a communication with the assigned tasks
- ...

The defined tests try to capture the actual expectations of the customer about the functionality expressed in the user story and the related cases that the developers have to consider. User stories are not usage scenarios but a description of the functionalities from the point of view of the customer.

3.1.7. Relationship with the customer

In a project where the requirements collection is performed through the collection of user stories there is a tighter connection with the customer compared to the connection created following a waterfall process [requirements collection, requirements analysis, coding, testing] where the communication with the customer is present mainly at the beginning during the definition of the usage scenarios and at the end during testing and deployment. Often, during the test phase the customer finds out that what was produced was not compliant with the expectations. This is because of the lack of feedback during the development of the functionalities of the software product.

During a project guided by user stories the customer should be involved in the work to provide continuous feedback enabling the developers to tune the product managing effectively the changes required by the customer.

After the definition of the stories in a specific meeting, the project manager and the customer decide the length of the iteration according to the needs of the customer and the estimation of the stories.

At the end of every iteration developers are responsible for providing working code and verified against the related acceptance tests.

The entire process depends on the correct and rigorous definition of the acceptance tests.



3.1.8. XPAME development process

XPAME is the agile development process proposed for Engisud, it includes the support for iterations and incremental development. During every iteration, all the phases of a traditional (waterfall) development process are implemented. Moreover, also inside a single iteration phases do not have to be organized in a sequential and predetermined way. It is possible to go back to a previous phase at any moment making the process *adaptive*. Obviously, two or more phases can be executed in parallel since, for instance, during the tests it is possible to plan an iteration, a release, or estimate user stories.

Figure 7 shows XPAME, the development process proposed for Engisud and the flow of events. The ones with a light blue background are prerequisites for the others, while the ones with a green background are not. For instance, it is not possible to define a release or iteration plan without the estimation of the user stories, while it is possible to define iteration plans without a release plan.

The showed phases are the following:

- **Collection of the user stories.** The functional area together with the customer participates to *user stories elicitation* sessions in which user stories and acceptance tests are defined. Acceptance test are very important because they are the bases for writing the code. The description of the stories and the related tests is described in the sections 3.1.2 and 3.1.3. It is very important that the development of the stories is supported by technicians of the technical area since they have the experience required for inserting *notes* useful for the development team. If possible, it should be better that representatives of the developers are involved as well in the definition, since the oral communication is very important in such approaches.
- **Estimation of the user stories.** Stories are estimated through a Wideband-Delphi session in which developers propose estimates at 50% and 90% of confidence level. If estimates are very high (such as 5 times the team velocity), this is an epic that has to be split into several smaller stories. Therefore, after a first session, the splitting is performed followed by the estimation of the splits. Estimation is very important since allows the development of release and iteration plans.
- **Release Planning.** An high level release plan shows approximately when a set of estimated stories will be implemented. Through this plan it is possible to estimate the effort that will be spent for a set of stories. Obviously, all the plans related to the shipment of a product is related to the team velocity. To calculate the team velocity, the developers have to provide an estimate of the story points they are able to implement in a working day during every iteration.

There are three methods for calculating the team velocity:

- Past data
- Perform an iteration and use the calculated velocity for all the iterations
- Forecast the velocity on the base of principles used for the estimation of the user stories

Using past data is not the best option since the values are strictly dependant on the skills of the developers and it does not consider the content of the user stories.

The execution of an initial iteration is the best solution when the members of the team have no experience about the development process in Engisud.

When the development team has experience about how to estimate velocity and is able to provide reliable estimates, the best approach is the forecast of the velocity on the base of the principles used for the estimation of the stories. This approach called *Forecasting velocity* is based on the assumption that on the base of their experience developers provide estimates of the number of story points they will implement in a working day during each iteration. This measure allow the calculation of the velocity in each iteration, how many working days are required for the implementation of a specific number of story points, and at which iteration the selected stories will be implemented.

This operation can be done only if there are resources assigned to the project. If this assignment is not done, the calculation is not possible.

During the release planning the customer, supported by a representative of the technical area, selects a set of estimate and coherent stories. In this way, the estimated number of working days required for their implementation is:

$$days = \frac{\sum_{j=1}^n story\ points_j}{team\ velocity}$$

The release plan presents is a long term view, therefore the estimate is highly approximate

- **Iteration Planning.** The Iteration Plan adds details to the Release Plan focusing only on a single iteration.

To plan an iteration the development team calls a meeting in which all the stakeholders should be involved: project managers, developers, and customers.

During the Iteration Planning the following sequence of activities is performed:

- Discussion of the single user story
- Definition of the tasks for the story (if needed)
- Assignment of the *responsibility* related to the development tasks.

During the Iteration Planning the customer sorts the user stories according to their priority. When all the stories have been discussed, developers estimate the accepted tasks.

The goal of this phase is to saturate the story points that the team can develop in a single iteration. On the base of the velocity of the team in the iteration (the iteration length is fixed during the definition of the iteration) the customer with the support of the technicians selects a set of valid and coherently ordered stories for the iteration.

$$\sum_j (Story\ points)_j \leq (Team\ velocity) \times (iteration\ length)$$

A set of user stories is valid when they present a conceptual link among them, while it is coherently ordered when the implementation order does not affect the validity of the set.

- **Test Driven Development.** People from the technical area together with the developers translate the acceptance tests into JUnit test cases. This is for implementing the Test Driver Development. This is the most important step since it is the mapping between the informal description of the requirements (user stories and acceptance tests) and the code. If the tests are well written they allow checking the code from the point of view of the customer. Tests have to be written by personnel with experience in translating informal technical



requirements. Unit tests defined for each developer will guide the development of the code since the goal of the developer will be the implementation of the code able to satisfy the tests. The set of the tests defined for each developer is a suite that will provide support to the refactoring activities.

- **Refactoring.** It is an activity required to improve the quality and the readability of the source code. Continuous integration activities are required to verify the compatibility of the different pieces of the developed code. The technical area has to develop *integration tests* as well. At present, the development process of Engisud states that tests have to be formalized into documents. This activity should be translated into JUnit test cases.

At the end of a cycle, if what is developed is a release, the functional area of Engisud. can consider the code ready to ship with a low number of bugs. However, the number of bugs is strictly related to the quality of the unit and integration tests.

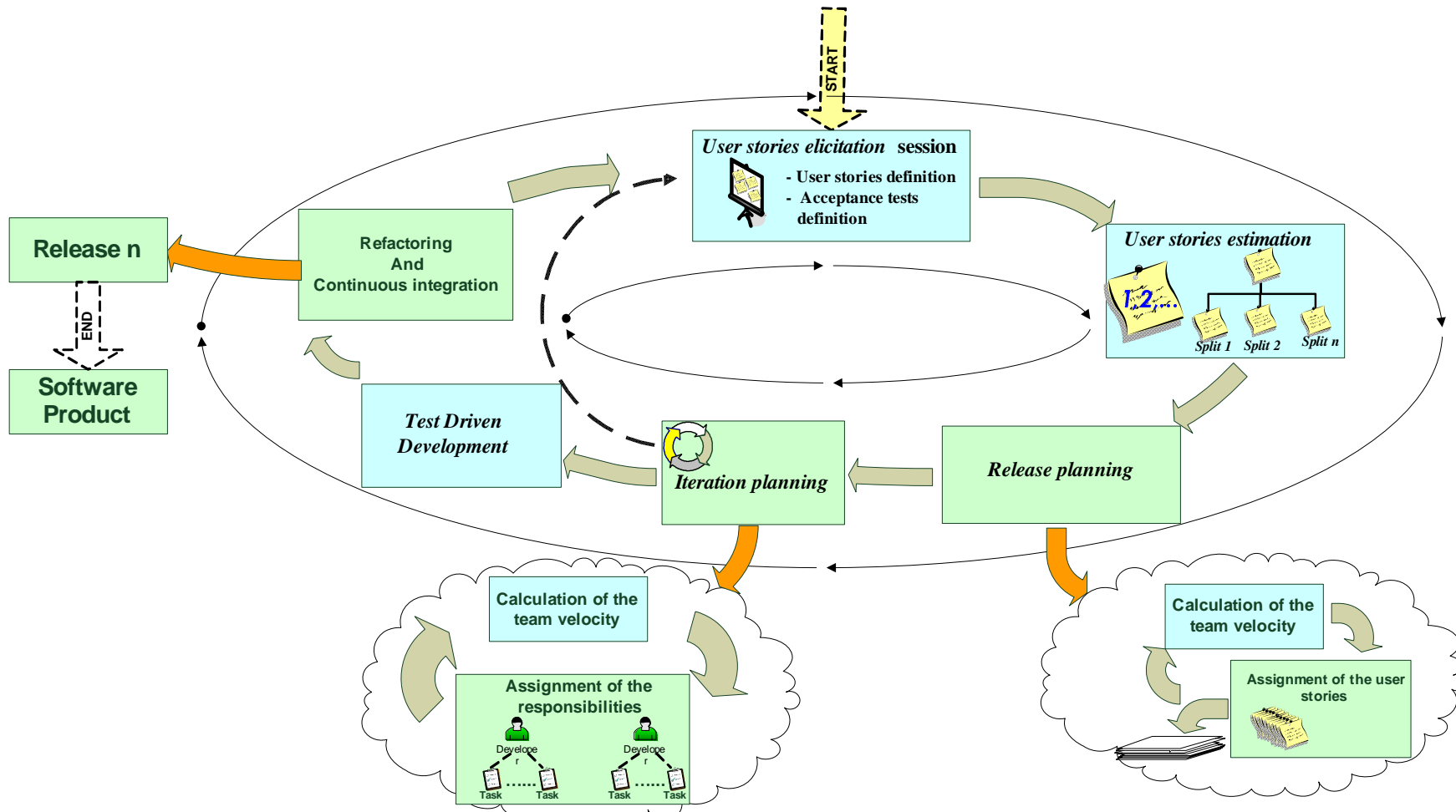


Figure 7. Life cycle of a project according to the agile approach proposed for Engisud

3.1.9. Tools to support XPAME.

The development process described in the section 3.1.8 is supported through two tools dealing with the management and implementation of a product:

1. **eXtreme Project manager:** a project management tool to guide the definition of requirements through user stories and acceptance tests. It supports an iterative and incremental development with the support of the short (iteration level) and long (release level) term planning through metrics like *team velocity*, *iteration length*, *story estimates*.
2. **Test case generator:** developed as a stand-alone tool and as an Eclipse plug-in, guides the developer in the implementation of the Test Driven Development. This tool has to be used by the tester. The input of the tool is the information related to the responsibilities of the developers and the assigned tasks. Further information related to the data and their flows are described in [4].

The high level representation of the development process is showed in Figure 8.

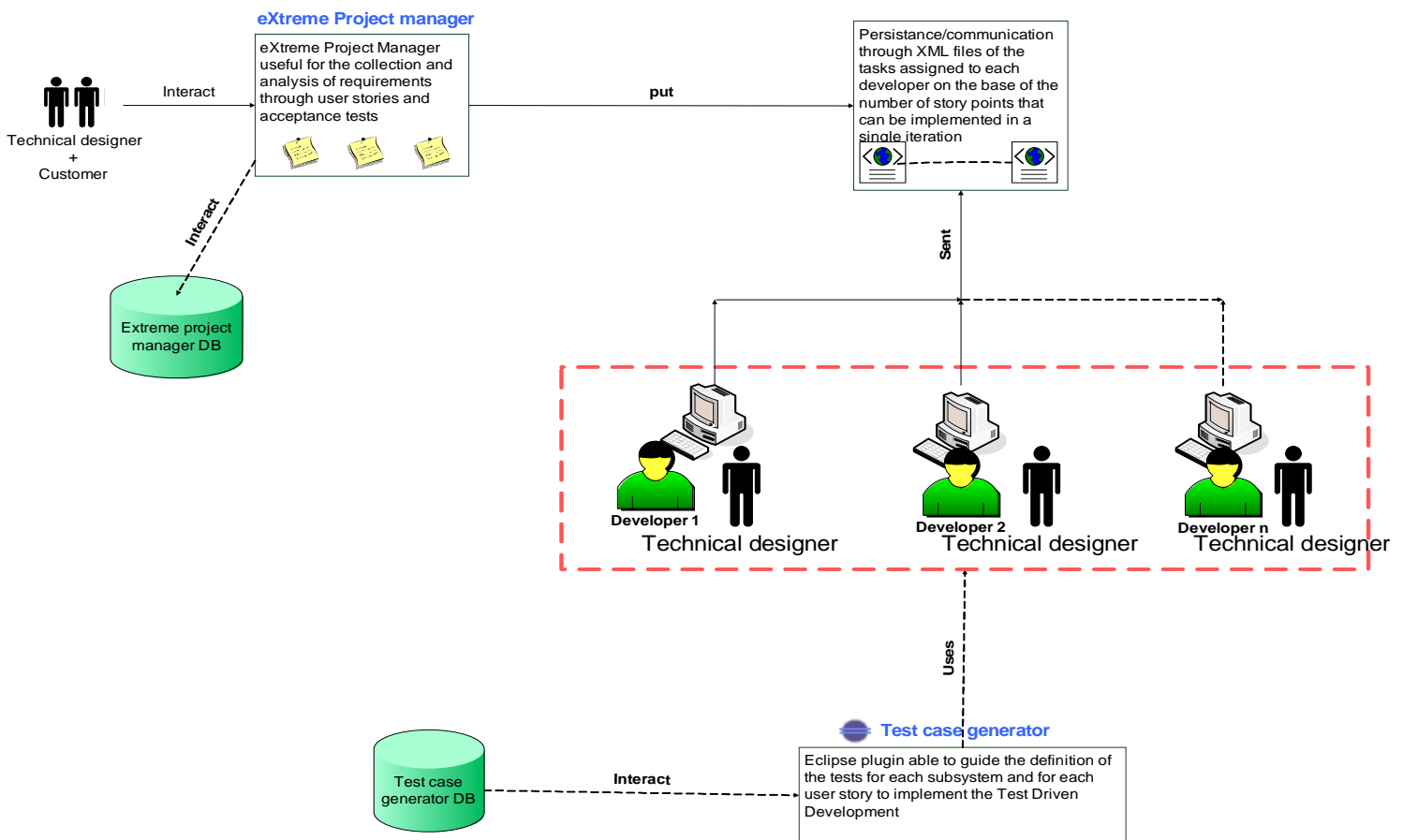


Figure 8. High level overview of the system

In the high level view of the architecture there are the following main components:



1. A stand-alone application: eXtreme Project Manager guides the functional analyst in the definition of the user stories (including tasks and subtasks) and the acceptance tests in collaboration with the customer.
2. The part that guides the TDD includes the definition of the tasks and entities related to the user stories, the generation of the test cases before the implementation of the code. Such macro-component can be integrated into the IDE used in the company (Eclipse) since it is a plug-in that is able to extend the standard functionalities and helps in the definition of the JUnit tests

Conclusions and future work

This document has described the XPAME development process, a collection of agile practices and processes thought for Engisud and based on critical issues presented during the interviews performed to representatives of functional, technical and development areas.

XPAME considers several agile best practices during the whole development process that would be deployed in the current development process of Engisud. Actually, Engisud is including done of these practices: unit and integration tests using automated test tools (i.e. JUnit), a close involvement of the customer during the whole development process, refactoring activity and continuous releases. For the future, Engisud, based on its needs, will adopt further agile best practices taken into consideration by XPAME in order to be as agile as possible.

Extreme project manager and Test case generator are useful tools supporting described agile practices so that they can be used for a general agile methodology adopting them.



References

- [1] JUnit framework web site, <http://www.junit.org>
- [2] Report of the meeting with the Engisud functional area responsible
- [3] Mike Cohn, “*User Stories Applied: For Agile Software Development*”, Addison-Wesley, March 2004
- [4] Giovanni Aiello, Marco Alessi, “eXtreme Project Manager, technical details”, October 2005
- [5] Giovanni Aiello, Marco Alessi, “Test case generator, technical details”, October 2005
- [6] Agile Manifesto for Agile software development, <http://agilemanifesto.org/>
- [7] Kent Beck, “eXtreme Programming Explained”, Addison Wesley, 2000