

Agile Requirements Engineering: Building tool support for XP

Contents

1.Introduction	3
2.Requirements Engineering	4
2.1.Requirements Engineering Process	4
2.2.Requirements Management Process	5
3.Extreme Programming	8
4.Related Work	11
4.1.Requirements Engineering in Extreme Programming	11
4.2.Challenges for Requirements Engineering	13
4.3.Existing solutions	14
5.Requirements Engineering approach for XP	15
5.1.Requirements Engineering Process	16
5.2.Integrated Development Environment	18
5.3.Proposed Tool	18
6.Discussion	21
7.Conclusion	21
References	22

1. Introduction

Agile methodologies have gained a lot of attention lately in software engineering industry. They promise more value for money and more chance of success for projects. Extreme programming (XP) is one of the most popular of these agile methods.

Management of user stories in requirements engineering viewpoint has got very little attention in the XP literature [1,2,3]. Requirements management and change management are perhaps those tasks that developers of XP method felt too bureaucratic or heavy weight to be included in XP. In current XP process user stories are written in (CRC) cards which are placed on the walls with their respective tasks cards. This is a very simple and powerful way of visualizing user stories when project team works in shared workspace. XP process doesn't exclude the use of automated tools for storing user stories, but they should be printed and put wall nevertheless.

Rigorous requirements engineering practices are important when it comes down to maintaining software [8,12]. Requirements management ensures that requirement are valid and system changes reflect changes in requirements and vice-versa [8]. Rigorous requirements management can be difficult. It is also effort consuming when traceability is maintained [8,12]. In a dynamic and fast moving project with an iterative process this effort consumption is increased too much that rigorous requirements engineering would be considered worthwhile. Agile methods and XP strives for efficient use of resources. It is therefore not good if major proportion of resources have to be used for management tasks. Tasks that do not deliver any concrete results are often found unpleasant for developers to do. Those tasks, however, have great value when, for example, effects of changes to system needs to be assessed [12].

This paper attempts to define way of executing the requirements engineering tasks in XP that are required for proper software engineering. Still that activity should not restrict the efficient execution of XP method. Requirements engineering procedures are studied from iterative and incremental process viewpoint. A minimized execution approach is defined together with a tool that supports this approach. This study concentrates on extreme programming methodology but principles that are introduced are likely to be applicable to projects that are using agile type incremental process model.

This study is composed as follows: First some requirements engineering and management concepts are introduced in chapter 2. Chapter 3 gives short introduction to XP and chapter 4.1 examines XP from requirements engineering point of view. Requirements engineering related

problems with XP are defined in chapter 4.2. Chapter 4.3 looks what other people have proposed as a solution for these problems and then chapter 5 introduces our approach to these problems.

This study uses few principal source literature and several other articles. Extreme programming related information is heavily based on Kent Beck's two books: *Extreme Programming Explained: embrace change* [1] and *Planning Extreme Programming* [3]. Requirements engineering principles are adopted from two books: Ian Sommerville and Peter Sawyer, *Requirements Engineering: A good practice guide* [8] and Gerald Kotonya and Ian Sommerville, *Requirements Engineering: Process and Techniques* [12].

This study also uses experiences gained in eXpert project in a Finnish research institute, VTT Electronics, during spring 2003. A team of four developers was acquired to implement a system (eXpert) for managing the research data obtained over years at VTT. The four developers were 5-6th year students with 1-5 year of industrial experience in software development.

2. Requirements Engineering

Requirements engineering is considered in traditional software engineering process to be the among the first phases of the project along with feasibility and business study.

2.1. Requirements Engineering Process

Requirements engineering includes requirements management activity which is an umbrella activity that covers whole project. Sommerville & Sawyer defines requirements engineering as follows [8]:

“A requirements engineering process is a structured set of activities which are followed to derive, validate and maintain system requirements document.”

Requirements engineering has thus three basic activities which are usually found in proper requirement engineering process: requirements elicitation, analysis and validation (Figure 1). Then there is a requirements management activity for supporting all of these activities [8].

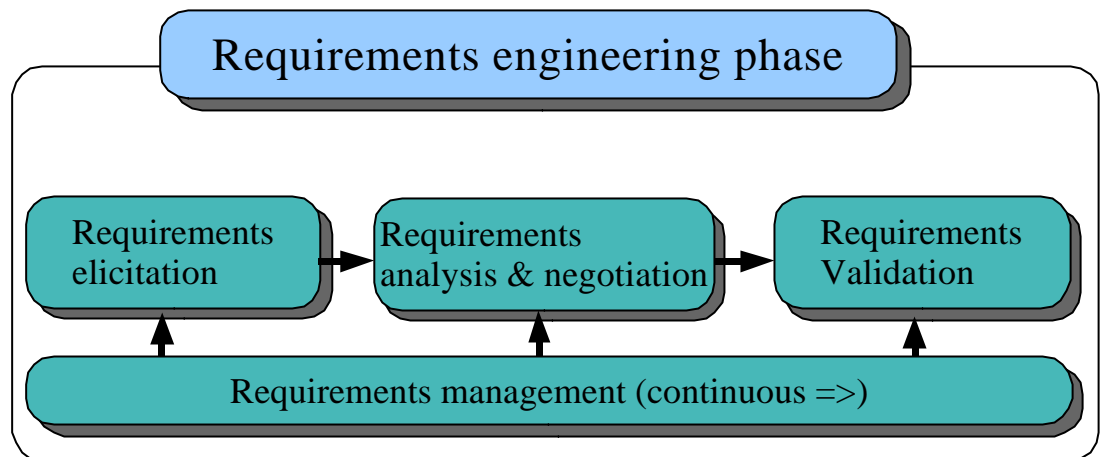


Figure 1 Traditional requirements engineering process

Requirements elicitation is an activity where requirements are discovered and recorded. In the requirements analysis, the requirements are analysed in detail and accepted by relevant stakeholders.

Requirements analysis and negotiation is phase where requirements are analysed in detail and involves different stakeholders. In this phase requirements are accepted for implementation. In requirements validation phase requirements are carefully checked for completeness.

2.2. Requirements Management Process

Sommerville and Sawyer (1997) and Kotonya and Sommerville (1998) define requirements management (RM) as a parallel support process for other requirements engineering processes. Furthermore Sommerville and Sawyer (1997) define principles concerning requirements management. Sommerville and Sawyer (1997) state that changes to the agreed requirements need to be managed along with relationships between requirements. Also relations between requirements documents and other documents produced during the systems and software engineering needs to be managed. According to these principles the main concerns of requirements management are change control and traceability.

Traceability can be divided into two subcategories, namely pre- and post-traceability. Pre-traceability involves requirements life before its inclusion in baselined requirements specification [7]. Post-traceability is defined to mean requirements life after inclusion in baselined requirements specification [7]. In this study we define that requirement to requirement traceability is post-

traceability. Pre-traceability means traceability from requirement to sources of requirement (client, law, standard, etc.).

Sommerville and Sawyer (1997) have proposed some guidelines for the requirements management that extend RM concepts to also cover identification, planning and automation topics:

Uniquely identify each requirement	Each requirement should be assigned unique identifier, which may be used to refer to that requirement (Guideline 9.1).
Define policies for requirements management	Define goals for requirement management, the procedures which should be followed and standards which should be used (Guideline 9.2).
Define traceability policies	You should define what traceability information should be maintained and how this should be represented (Guideline 9.3).
Maintain a traceability manual	Traceability manual includes the specific traceability policies used in a project and all requirements traceability information (Guideline 9.4).
Use a database to manage requirements	Rather than maintaining requirements in text documents, establish a requirements database and store individual requirements as entries (Guideline 9.5).
Define change management policies	Policies set out how changes are formally proposed, analysed and reviewed (Guideline 9.6).
Identify global system requirements	Global system requirements set out desirable or essential properties of the system as a whole and they are very expensive to change (Guideline 9.7).
Identify volatile requirements	List of volatile requirements should be maintained, that is, those requirements which are most likely to change (Guideline 9.8).
Record rejected requirements	Keep record of requirements which have been proposed and rejected after analysis and negotiation (Guideline 9.9).

MOOSE¹ inventory [11] combines these RM guidelines under four main activities as follows (Figure 2):

- **Requirements identification:** all guidelines which relate to the identification and storage of requirement items. Requirements identification focuses on the assignment of unique identifier and appropriate additional information for each requirement.
- **Requirements traceability:** all guidelines which relate to the requirements traceability. Requirements traceability (RT) refers to the ability to describe and follow the life of a requirement in both a forwards and backwards direction [7].
- **Requirements change management:** all guidelines, which relate to the requirements change management. Requirements change management refers to the ability to manage changes to the systems requirements [12]. It ensures that similar information is collected for each proposed change and that overall judgements are made about the costs and benefits of proposed change.
- **Requirements management planning:** guidelines, which relate to the planning and documentation of identification, traceability and change management activities as well as the definition of other RM goals, responsibilities and policies for a project. These practices define goals and procedures for RM, which should be followed in an organization. It provides means to select and define suitable RM procedures when considering RM for a project.

¹ MethOdOlogieS for Embedded Systems is ITEA project

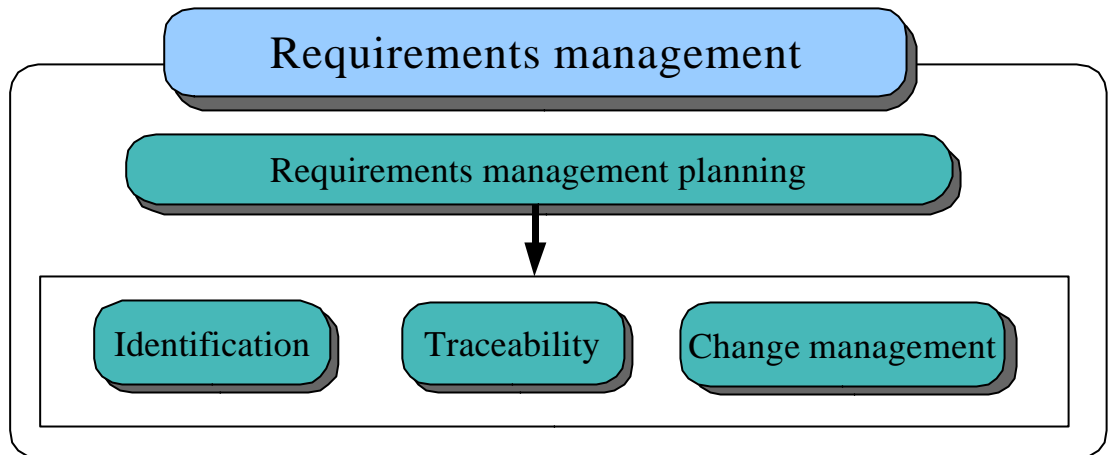


Figure 2 Main activities of RM

3. Extreme Programming

Extreme programming (XP) as a concept has emerged late 90's along with Kent Beck's book "Extreme Programming explained: Embrace Change" [1]. Along with XP many more of these "agile" methods (for an overview see [14]) have emerged such as Dynamic System Development Method (DSDM) [15], Scrum [19] and Crystal family of methods [20]. One objective of these agile methods is to lower cost of changing requirements.

XP addresses issues of changing requirements and their cost by simplifying management tasks and documentation. XP uses iterative and incremental software process in relatively short cycles. Traditionally this type of approach would yield increased management overhead because management activities related to ending and starting iteration have to be executed for every iteration but these are minimized in XP process.

XP introduces many practices but two techniques which are characteristic to XP are pair programming and test driven development [1]. XP may first seem just an acronym for ad-hoc development, but when practices are operationalized it soon becomes evident that XP is actually a rather strict methodology [16]. In fact Paulk (2001) argues that XP fulfills CMM level 2 key process areas and even many level 3 key process areas. XP is not, however, a silver bullet and not applicable to all projects. Also loosening of requirement management practices have raised some concerns [5]. In the following the principal practices and approaches of XP are listed [1,3]:

Practice	Explanation
Small releases	XP project is divided into small releases of maximum of two months. System designed for one release at the time. Two months release can be divided into several iterations. According to experiences gained in eXpert project duration of one iteration should be no less than two weeks if new features are introduced to system [13].
Incremental development	XP projects are conducted in an iterative process where each release produces working product which is tested and can be deployed.
Pair programming	Development is done in two programmers team where one pair shares one computer. It is argued that quality of produced code is much better than in solo programming and consumed effort in relation to results is not in fact twice as much.[17]
Continuous integration	Teams integrate their code often in one integration machine. It is convenient to use version management tool's repository for integration and then separate integration machine is not obligatory. Version management tool should use optimistic concurrency; in other words it should be non-locking [18] (i.e. CVS).
Automated acceptance test	Acceptance tests should be automated with some appropriate script language. Customer writes these but the development team should provide assistance if customer is not able to construct acceptance tests by himself. Test scripts corresponds to user stories and validate them. All scripts are run at the end of each release before product is handed to customer.
Test driven development	When production code is written, unit tests for the code are written before it. This forces developers to think module interfaces and design before implementation. Test can be written with some testing frameworks such as xUnit (JUnit for Java).
Shared workspace	All developers should be located in same open workspace. This simplifies communication and creates productive atmosphere.

Practice	Explanation
On-site customer	Customer should be always present in the same workspace as developers. This is a practice that helps facilitating communication about changing and ambiguous requirements.
40-hour weeks	No overtime should be worked or not at least for two weeks a row.
Planning game	Planning game is a phase in XP development when requirements, that is stories, are elicited, estimated and selected for release.

During planning game, customer writes stories, which the developers estimates and customer then subsequently prioritizes. Subset of stories based on priority and size is then selected for each iteration. Developers then divide stories in into tasks and give an estimate for each task. Estimating user stories is difficult in other than very coarse level, on the other hand estimating tasks is much more easy and accurate because tasks are defined in more detailed and concrete level. In the literature [1,3] these two steps which involves estimation are divided into two separate phases planning game and iteration planning. At least in a smaller project it is advisable to merge these two phases in order to produce more accurate estimates. This approach was used in XP experiment (eXpert) in VTT Electronics [13] with good results.

In the process that was used in eXpert client wrote stories as usual. Development team then deduced tasks for stories and estimated tasks. Size estimates of tasks that relate to some specific story are then summarized and allocated to story. Story estimates become more accurate this way. Duration of planning game is of course increases significantly but time for planning as a whole is the same because same estimates have to be done in anyway and iteration planning is not needed. Keeping planning game and iteration planning separate is well justified when release contains several iterations and only portion of stories that are assigned for release are composed to tasks and estimated.

4. Related Work

There is increasingly more work done for requirements engineering aspect of extreme programming. Original approach of extreme programming for requirements is felt inadequate in many cases and some modifications for XP process are represented.

4.1. Requirements Engineering in Extreme Programming

Requirements engineering is divided into three practices in the extreme programming process [1] planning game, release planning, iteration planning. Usually project has several iterations for each release, but release can also be in every iteration [3] depending of scope of project.

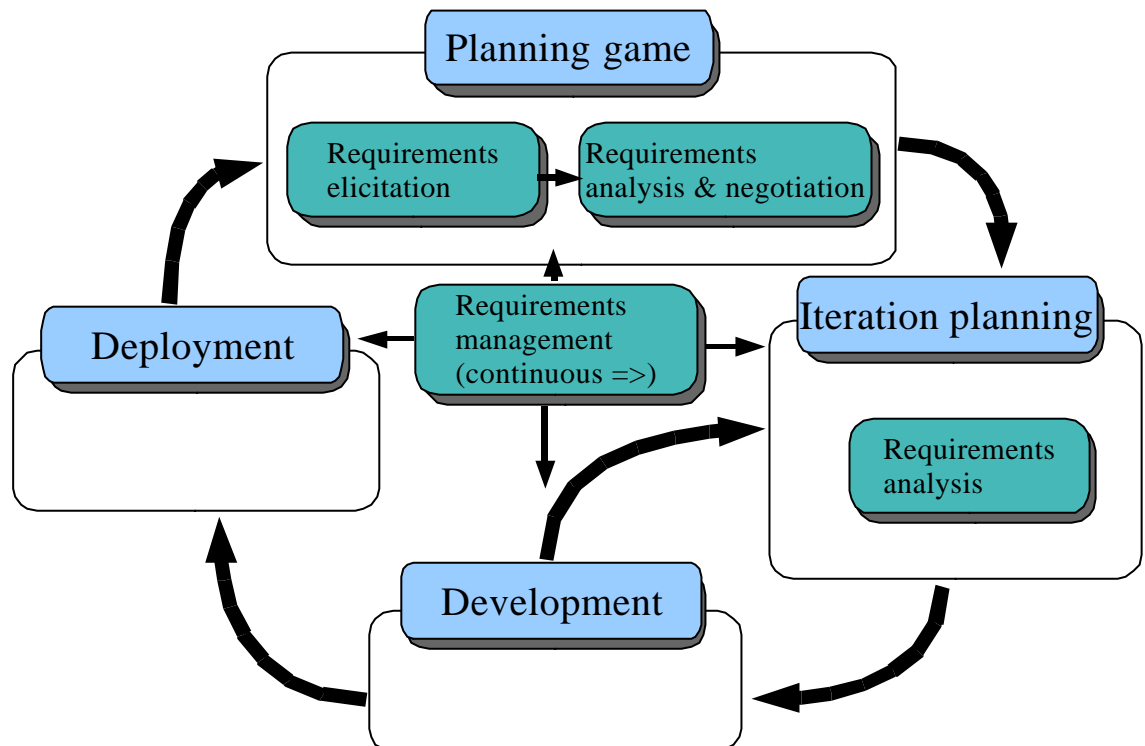


Figure 3 Requirements engineering in XP

Requirement engineering activities introduced in the section 2.1 can be found in the XP process. Requirements elicitation activity is done during planning game and responsibility of this activity lies largely on clients shoulders. Customer writes stories before planning game or during planning game. Development team of course helps in this task by providing their expertise of feasibility and such. Requirements analysis and negotiation is divided between planning game and iteration planning. Requirements negotiation is done mainly in planning game by prioritizing requirements. Requirements analysis is done partially in planning game when development team estimates cost of story. More detailed analysis of requirements is conducted in iteration planning where tasks are

defined. Requirement validation activity is not included in XP practices in any formal way other. XP practice of on-site customer is insurance that when missing or contradicting requirements are found their effects can be swiftly analyzed and iteration can be steered accordingly. This also relates to requirements management activity of change management. Requirements management activity affects all phases of project in terms of change management and traceability.

User story can be thought as a high level requirement or a user requirement as it is users conception of single thing what system should provide.

4.2.Challenges for Requirements Engineering

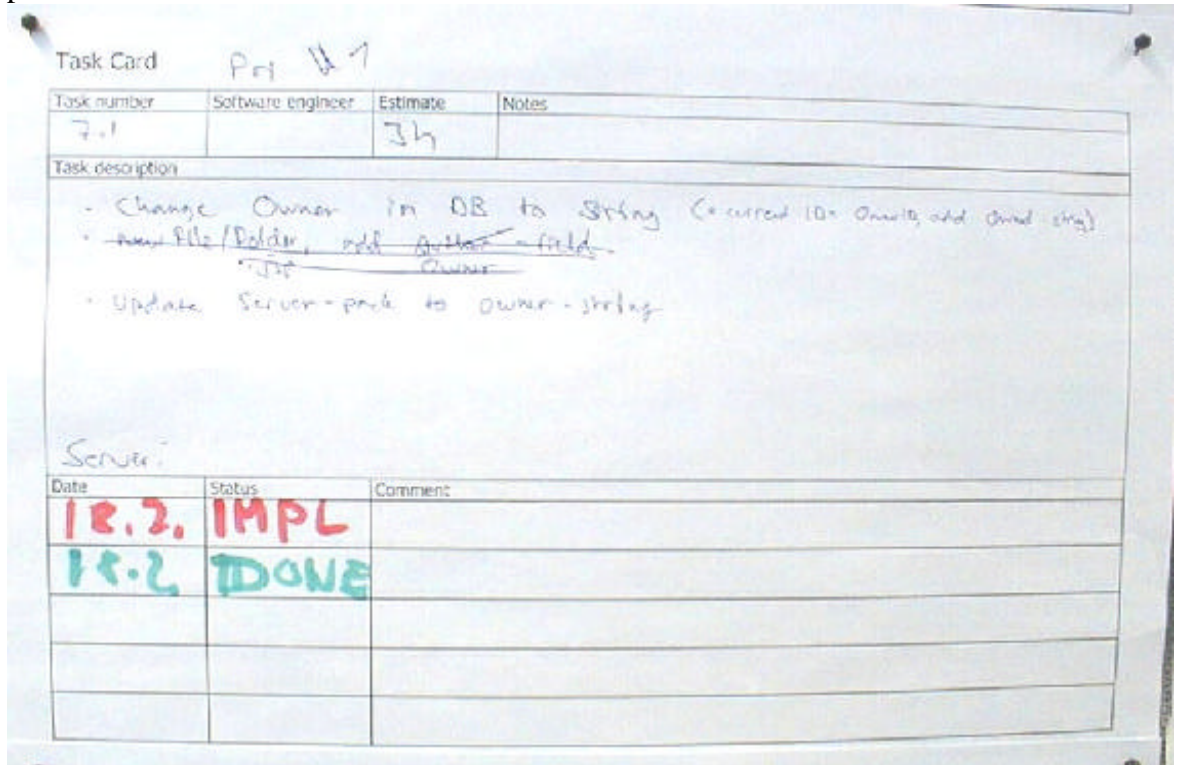
XP has a principle of travel light [1], which means that user stories that have been implemented in previous releases are discarded [3]. This is not necessarily good practice [6]. Idea is that automated acceptance tests provide necessary information about implemented functionality [3]. It is also argued that information value of acceptance test to customer is not necessarily as good as plain English user stories [6].

Even though techniques proposed in literature [1] is such a flexible and illustrative, it does not necessarily answer all needs that project might have. If stories should persist as is proposed [6], then also relations that they have should persist. If stories would be stored just to paper as story and task cards, they would be bound to get lost in some point of project as wall is cleared for new iteration. Needless to say that so would relations that they may have. So we need process that addresses these issues and which is backed by automated tool. These type of processes and tools exist already, but those processes are heavy and generate too much overhead in dynamic and fast moving projects such as XP project.

User stories are very simple technique and consequently tools that support it should be also simple and easy to use. Otherwise developers rather use pen and paper. So we need to find middle road that is light enough yet provides necessary rigor.

One other problem is that type of information stored in story and task cards varies from project to project. Card used in Chrysler C3 project [1,2] are somewhat different and more complex than in eXpert project [13] at VTT. Even though cards were simple, not all of their fields were used. In this small co-located project it made no sense to allocate task to some specific individual so “software engineer” -box was left empty in all but first iteration task cards (Figure 4). On the

other hand task card could have contained “priority” field because all tasks and stories were prioritized.



Task Card Pri 47

Task number	Software engineer	Estimate	Notes
7.1		3h	

Task description

- Change Owner in DB to string (current ID= Owner, add Owner string)
- ~~new file/folder, add Author-field~~
~~IT~~ ~~Owner~~
- Update Server-path to owner-string

Server:

Date	Status	Comment
18.2.	IMPL	
18.2	DONE	

Figure 4 Task card of eXpert project

Because needs of a project differs, process and tool should be adaptable to variety of different kind of projects and yet be simple and easy to use. Tool should not require strict process. Use of tool should not cause overhead for process or overhead should be small in relation to added value of information preserved in the tool.

4.3.Existing solutions

Nawrocki et al. propose modifications to XP process which includes making light requirements documentation [5]. One of their concerns is that on-site customer that has authority to make business decisions is rare luxury and makes XP project expensive from customer point of view. On-site customer necessarily does not cost too much for client if customer is able to work normally in a project room while project team does not need him as was in eXpert project. Nawrocki, Jasinski and Walter propose that tester is made responsible for requirements documentation and this person is called **tester/analyst**. They also give several pointers for documenting requirements:

- Do not be afraid to use advanced tools

- Organize the requirements into multiple layers
- Make the collected requirements available to the team and to the customer through the web
- Carefully choose attributes for your requirements
- Transfer all enclosures provided by customer to electrical form.

Use of automated tool for user story management is advocated. Currently there is distinct lack of automated tool supporting for user story management. MILOS tool for distributed XP [4] is one of the very few RE tools that are developed for XP. Organizing stories into multiple layers may complicate planning game and iteration planning unnecessarily much. One might also question the value of requirements being available in the web if customer sits on same or next door. Some sort of summarized release plan of functionality and schedule may be in order to catch public attention if such is relevant.

Traditional requirements engineering tools can be used for this, but they provide more functionality that is needed for the purposes of XP. Tools that have lots of unnecessary functionality are often hard to learn because of increased complexity of tool.

Breitman and Leite [6] present a framework for managing user stories. Their approach is based on applying principles of scenarios based development to user stories. While their approach is interesting it lacks throughout support for tasks. Episodes in their framework corresponds to tasks in XP. Framework promises traceability from stories to source code, but tasks are not explicitly addressed. Tasks have also important information embedded which should be preserved just as well as stories. Including tasks in tool support should not complicate usage any more than including the user stories.

Frauke Paetsch has also compared traditional requirements engineering approaches to agile software development. Paetsch also determine how agile methods can benefit from traditional engineering methods and shows how one RE technique can be integrated in a tool that supports agile software development that is distributed over the internet. [21]

5. Requirements Engineering approach for XP

Modifications presented for XP relate more to tool support than in process itself. XP process for RE is effective for at least for smaller projects. Lack of requirements documentation is considered

to be one of the weaknesses of XP as well as one of the greatest strengths. With tool support these weaknesses can be strengthened without hindering agility of the process.

5.1. Requirements Engineering Process

User stories are managed manually during iteration traditional process. As have been seen in previous chapter, it would be often preferable to have these stories stored persistently in electrical form. Manual technique is however very elaborate and if stories and tasks are managed electronically they should be also printed and put to wall for communication purposes if necessary.

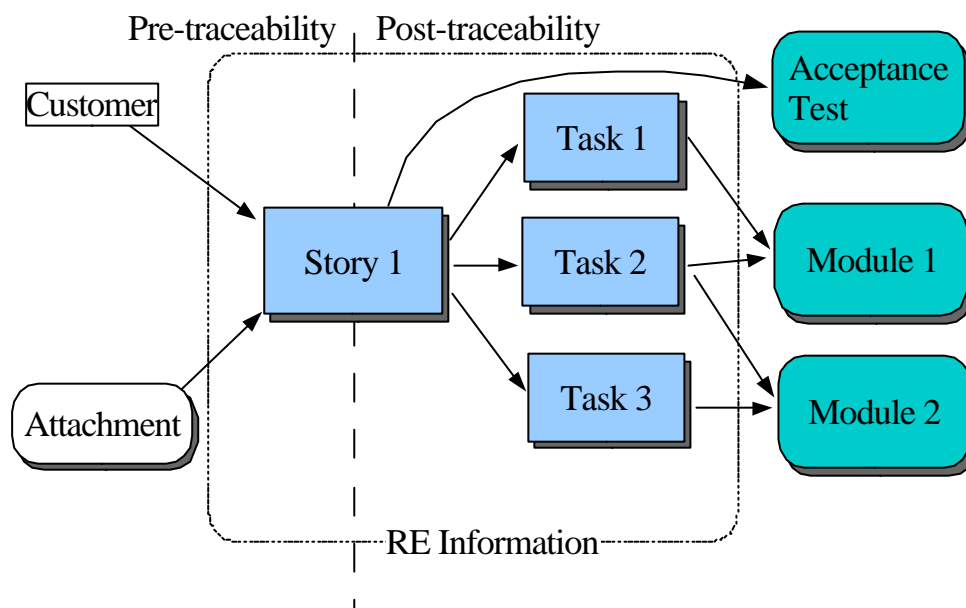


Figure 5 RE information in XP process

Stories can be considered as user requirements but they often are in very coarse level. Tasks break stories into several phases that needs to be accomplished for given story. Different objects that relate to requirements and their relations are illustrated in Figure 5. White sharp cornered boxes are stakeholders. Arrows represent traces or links between items and fine dashed box represent information that is stored in RE tool. Sharp cornered and shaded boxes are objects that contain RE information. Rounded corner and coloured boxes are implementation or design objects. There also can be hierarchical relations between stories which are also post-traceability relations.

Requirements engineering process should go as is presented in literature (Chapter 4.1) for planning game if one release contains several iterations. Stories are written by customer(s) with the help of development team. Only modification is that stories are written to automated tool. If

customer provides source documentation for stories, they are also stored into tool environment. All stored items should be under version control. Each user story can be linked to source of user story which can be some document or most often customer. Customer should prioritise all stories and with the help of development team they should identify and select core set for the first iteration.

Core set may not be all most high priority stories but such a set that it is as valuable to customer and forces development to form as complete architecture for system as possible. This is done because then in subsequent it becomes more unlikely that whole system structure needs to be modified. [1]

Development team may then evaluate stories and put a price (estimate effort) for them if there are several iterations before first release. Next, the team selects stories for the first iteration and decomposes them into tasks and estimates effort for these tasks. Tasks are also written to automated tool. Tasks are linked to stories and sum of time estimates of these tasks are then updated to corresponding story. If there is only one iteration per release development can decompose selected stories to tasks and evaluate them before stories are estimated. This assures a greater accuracy for initial story estimates. Finally stories and tasks are printed and put to wall on display.

When the team starts coding and selects tasks they also maintain links between tasks and code by selecting related implementation components in tool environments and by assigning them for task. When stories are ready (all tasks related to that story are ready) and acceptance tests for that story is written link is made between story and acceptance test script.

Maintaining traceability is challenging tasks for any software project. This is further emphasised in iterative and incremental process where refactoring is common practice later on project. Changes are that when code is heavily refactored links between code and tasks are broken. Therefore links between tasks and code need to include version of code component, this way link remain valid all the time even if code changes. Stories are linked to acceptance tests and those tests need to run all the time in all iterations so those links remain valid throughout the project.

5.2.Integrated Development Environment

Tool operating environment is Eclipse integration framework (www.eclipse.org). Eclipse is open source initiative supported by all major software engineering tool manufacturers and is based on

tool developed by IBM. Eclipse already supports extreme programming adequately well in a sense that the required testing framework and shared code base tools (JUnit and CVS) are integrated as a default. Integration of tools such as CVS (or PVCS or SourceSafe) indicates that step to adopt better configuration management practices is not too steep.

5.3. Proposed Tool

In tool framework which is proposed here, user stories are managed in a database and can be printed for "on the wall"-presentation. Tool provides simple version management for user stories with the possibility to make baselines and browse version history. Tool also stores tasks in the same way that it stores user stories. Attributes of stories or tasks are almost entirely up to user to define. State of tasks are explicitly defined by the tool and the tool is able to gather statistical data from tasks and stories like estimated time vs. actual effort. This provides support for other project management tools that may use this plug-in. Tool also exports stories and tasks into single documentation and is able to trace tasks into related code and into Javadoc documentation if such is relevant

Here is proposed two related information model, one meta-model (Figure 6) which defines information structure of concrete model and concrete information model which stores actual information. Figure is ER like notation where boxes represent entities and ovals represent attributes. Requirement entity can be either task or story in extreme programming context and all requirements have id and state parameter as well as any number of user definable attributes.

Possible state attribute values are:

Defined	Defined state is initial state which is when story or task has been written.
Implementing	Implementing state is activated when task has been assigned to some developer and story is in implementing state when some of its tasks are implementing state.
Done	Done state is activated for task when task has been done. Story is in done state when all its tasks are also in done state.
Postponed	Postponed state is activated for story if it, or one of its tasks are postponed to some subsequent iteration.

Defined	Defined state is initial state which is when story or task has been written.
Removed	Removed state is activated when story or task is deleted from repository. Such can happen for task when it is noticed to be futile or for story when it is found to be nonfeasible.

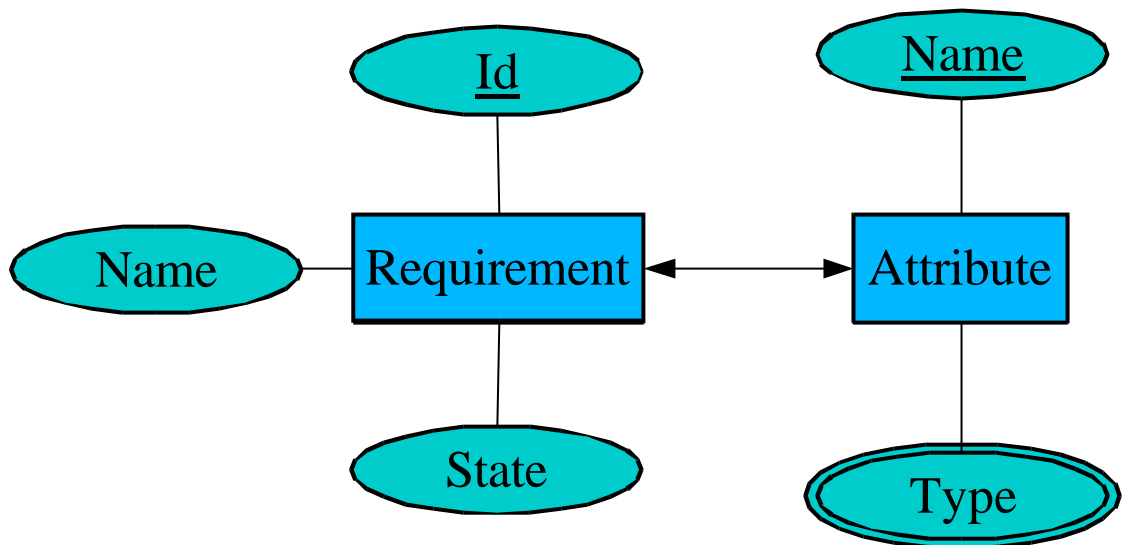


Figure 6 Meta-model of tool

Attribute entity's type attribute defines whether attribute is integer, string or what ever database related type, name attribute defines attribute entity's name. Meta-model is used to form information (database tables) structure for one specific project. There can be several type of requirements all which has unique id and they all have similar relations as Task and Story have in Figure 7. Information presented in meta-model is also stored in tool specific database. Whenever new project is created tool reads what Requirements types and parameters are defined in meta-model database and constructs project specific database based on those.

In Figure 7 there is presented data model of proposed system as ER diagram. Model borrows some elements from Breitman's and Leite's framework [6], but composition of stories and task is different. Also information can be stored to relational database or in XML file. Traceability is implemented by storing information about traced items into database. Item entity has multi value attribute item type, which can be one of these: Task, Story or File. Actual traces are stored in link tables which is constructed from N to M relationship between item and story or task. Location

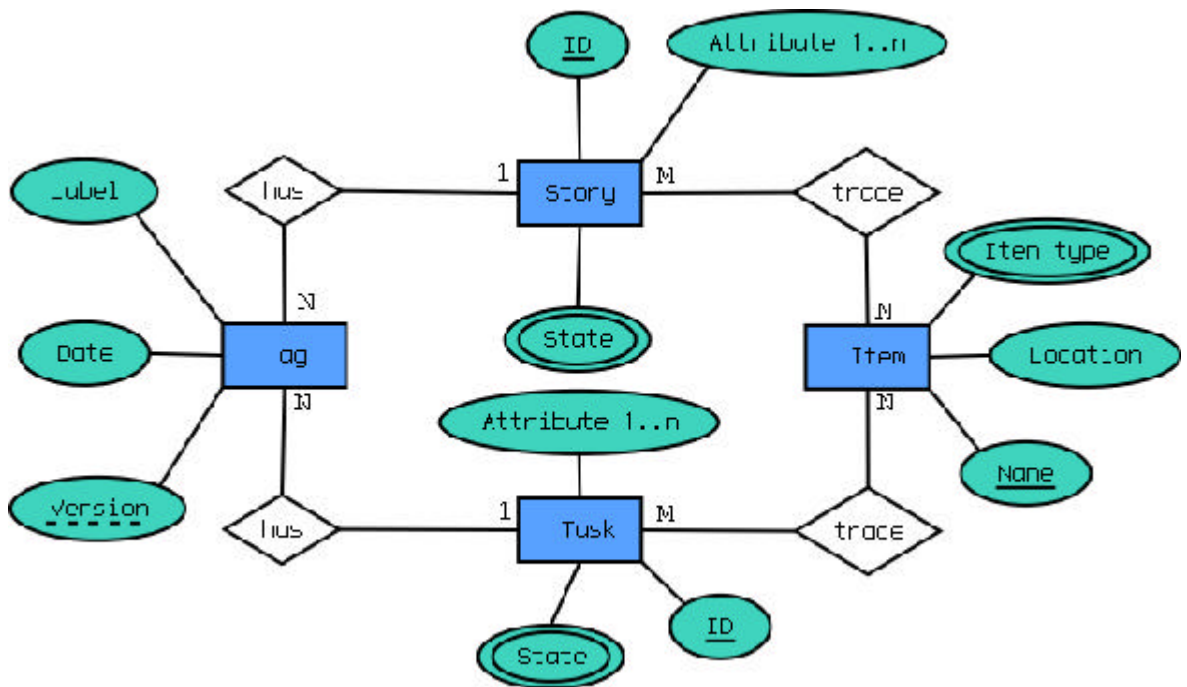


Figure 7 Entity Relationship (ER) model of proposed solution

attribute stores Items identification which can be for example path in Eclipse workspace or Story ID. Item entity and its attributes are managed by tool and used do not need to worry about it.

User Story and Task tables contain information of stories and tasks. Only ID field and multi-value state field are mandatory. ID field is combination of Requirement types ID in meta-model and incremental ID of task or story. This way tasks and stories always have unique identifier. Other fields are user definable and used when needed. Pre-traceability information such as source of story, rationale, etc. can be stored into these user defined attributes.

Tag table stores information relevant to version management. Version management in system is linear and on only the last version is preserved. So the purpose of tag table is gather and provide history information. Thus accessing past versions of user stories is not possible and therefore branching is not possible. Entry is added to Tag whenever user story or task is modified. Tag stores date and version number of related requirement. Version number is weak key and is used together with item ID to identify single tag. Tag also has label parameter which can be used to identify baselines. Baselines are parsed to separate file which can be placed onto wall.

By storing this information it is possible to construct tool that gives rigor to requirement engineering in extreme programming. Information that is stored is minimalist yet essential information is captured. Tool can be customized for different kind of projects, for example tool does not say that tasks should be stored in tool or that stories should be named as stories and not as requirements. Integrating tool to development environment makes tool more easily accessible

for developers. Keeping functionality minimalist makes learning curve steep and tool can be easily adopted. This helps to reduce possible overhead which is generated by more rigorous requirement practices.

6. Discussion

Proposed process and tool is yet to be tried in pilot project, but the need for strengthening RE aspects of extreme programming is obvious.

Role of nonfunctional requirements is left open. XP does not address these requirements other ways than oral agreement between customer and development team. This aspect narrows applicability of extreme project. It should be fruitful to study extending XP practices even further to cover these aspects as well.

Tool that is produced as result of this study will be tested to see if these introduced practiced indeed give rigor for requirements engineering in XP and whether or not they increase management overhead.

7. Conclusion

Approach represented in this paper extends extreme programming practices to have better coverage of requirements engineering issues. This is achieved by strengthening especially requirements management practices of XP. In order not to increase management overhead, process is kept light and tool support is introduced. Process proposed here resembles original XP approach in great extend with the addition of tool support for managing requirement engineering related information. Basic XP practices and process remains the same as in traditional XP process. Tool support makes persistent requirements documentation possible and accessible. Tool also gives rigor for capturing traceability information that would be otherwise lost. Change management aspects of XP are not modified because it was felt essential for XP that change management is not slowed down by bureaucracy involved in more traditional process approaches.

Here requirements management is discussed in the context of extreme programming, but it does not limit the use of tool that is represented here. Tool is designed to be so adaptive that it can also

be used in project that employs other method than XP. Tool environment sets some limitations for use of tool as it works only in Eclipse integration environment. Eclipse itself is such a extensible that it suits for many different kind of projects and supports variety of platforms.

References

1. Beck, K., Extreme Programming explained: Embrace Change, Addison-Wesley (September 1999), ISBN: 201-61641-6.
2. Jeffries, R., et al., Extreme Programming Installed, Addison-Wesley (June 2001), ISBN: 201-70842-6
3. Beck, K., Planning Extreme Programming, Addison-Wesley (October 2000), ISBN: 0-201-71091-9
4. Maurer, F., Martel, S., Process Support for Distributed Extreme Programming Teams, ICSE 2002 Workshop on Global Software Development.
5. Nawrocki, J., et al., Extreme Programming Modified: Embrace Requirements Engineering Practices, 10th IEEE Joint International Requirements Engineering Conference, RE'02 Essen, Germany, September 2002.
6. Breitman, K. K., Leite, J. C. S., Managing User Stories, 10th IEEE Joint International Requirements engineering Conference, RE'02 Essen, Germany, September 2002.
7. Gotel, O., Finkelstein, A., An analysis of the requirements traceability problem, Proceedings of the First International Conference on Requirements Engineering, 18-22 Apr 1994
8. Sommerville, I., Sawyer, P., Requirements Engineering: A good practice guide, John Wiley & Sons, 1997, ISBN: 0-471-97444-7
9. Wagner, L., Extreme Requirements Engineering, Cutter IT Journal, vol.14, no.12, December 2001
10. Ambier, S. W., Agile Requirements Modeling,
<http://www.agilemodeling.com/essays/agileRequirements.htm> [2 May 2003]

11. MOOSE WP2 Inventory: Literature Study, Report [D2.1.1](#), MOOSE (Software Engineering MethOdOlogieS for Embedded Systems) Consortium (ITEA 01002 Project), 2003
12. Kotonya G., Sommerville, I., Requirements Engineering: Process and Techniques, John Wiley & Sons, 1998, ISBN: 0-471-97208-8
13. Abrahamsson, P., Extreme Programming: First Results from a Controlled Case Study, to appear into Euromicro 2003, 2003
14. Abrahamsson, P., et al., Agile software development methods: Review and analysis, VTT Publication 478, Espoo 2002
15. Stapleton, J., DSDM Dynamic System Development Method: the method in practice, Addison-Wesley 1997
16. Paulk, N.C, Extreme programming from a CMM perspective, IEEE Software , Volume: 18 Issue: 6 , Page(s): 19 -26, Nov/Dec 2001
17. Williams, L., et al., Strengthening the case for pair programming, IEEE Software , Volume: 17 Issue: 4, Page(s): 19 -25, Jul/Aug 2000
18. Koskela, J., et al., Improving Software Configuration Management for Extreme Programming: A Controlled Case Study, ??.
19. Schwaber, K., Beedle, M., Agile Software Development With Scrum, Upper Saddle River, NJ, Prentice-Hall, 2002
20. Cockburn, A., Agile Software Development, Boston, Addison-Wesley, 2002
21. Paetsch, F., Requirements Engineering in Agile Software Development, Diploma Thesis, Fachhochschule Mannheim, Informationstechnik, (March 2003)