ISSN:0975-9646

G. Sivanageswara Rao et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 4 (1) , 2013, 73 - 76

Extreme Programming Verification using Unified Process

[1]G. Sivanageswara Rao [2]Ch.V.Phani Krishna [3]Dr. K.Rajasekhar Rao

[1,2] Associate Professor, [3]Dean S&F welfare
[1,2,3]KL University

Abstract--The movement has received much attention in software engineering. Established methodologies try to surf on the wave and present their methodologies as being agile, among those Rational Unified Process (RUP). In order to evaluate the statements we evaluate the RUP against Extreme Programming (XP) to find out to what extent they are similar end where they are different. We use a qualitative approach, utilizing a framework for comparison. We conclude from the analysis that the origin and business concepts of the two – commercial for RUP and freeware for XP – is a main source of the differences. RUP is a top-down solution and XP is a bottom-up approach. Which of the two is really best in different situations has to be investigated in new empirical studies

I. INTRODUCTION

The agile movement has appeared the last years as an alternative direction for software engineering [1]. Among the agile methodologies, Extreme Programming (XP) is the most well known [2]. In the current agile boom, many established software engineering methodologies try to present themselves as being agile. The Rational Unified Processes (RUP) [16] is among those, providing "plugins" to RUP for eXtreme Programming1. Thereby they offer a downsized version of RUP, which is stated to be lightweight, agile style. Both methodologies share some common characteristics; they are iterative, customer-oriented and role-based. RUP is generally not considered agile; rather it is criticized for being too extensive and heavyweight [21]. RUP comprises 80 artifacts while XP only stresses the code. RUP has 40 roles while XP has five.

Using a modified version of a standard question framework, we investigate similarities and differences between RUP and XP. The paper is outlined as RUP and XP briefly, as well as the research methodology.

II. RELATED WORK

Rational Unified Process

Rational Unified Process (RUP) is a development methodology, developed and marketed by Rational Software, by now owned by IBM. The first release came in 1998 and was a result of cooperation between Grady Booch, James Rumbaugh and Ivar Jacobson [12]. RUP is a general methodology that needs tailoring to specific organizations' and projects' needs.

The core values of RUP are [16]

- Use case driven design
- Process tailoring
- Tool support

The process is use case driven, and the use cases constitute the basis for other elements in the development process. The practical work in RUP consists of the following main items:

- Develop software iteratively
- Manage requirements
- Use a component-based architecture
- Model the software visually
- Verify the software quality continuously
- Manage software change

The RUP methodology is presented using four primary modeling elements:

- Roles – *who* is doing what
- Artifacts – *what* is produced
- Activities – *how* the work is conducted
- Workflows – *when* a task is conducted

To manage a software project, some kind of a project management model is needed, mostly of a stage-gate type [5]. This is integrated into RUP.

III. RELATED APPROACH FOR EXTREME PROGRAMMING(XP)

Extreme Programming (XP) is a lightweight development methodology, which stresses teamwork, communication, feedback, simplicity and problem solving [2]. XP consists of a set of software development practices, packaged into wholeness by Kent Beck and Ward Cunningham. Its roots are in the object-oriented community, specifically among SmallTalk programmers. XP is built on four values:

- Communication
- Feedback
- Simplicity
- Courage

Through communication within and outside the project, it is ensured that the right product is developed. Quick and frequent feedback provides abilities to correct the direction of the project. Simplicity means building the right product, not a product for possible future needs. Courage is needed to maintain openness and communication.

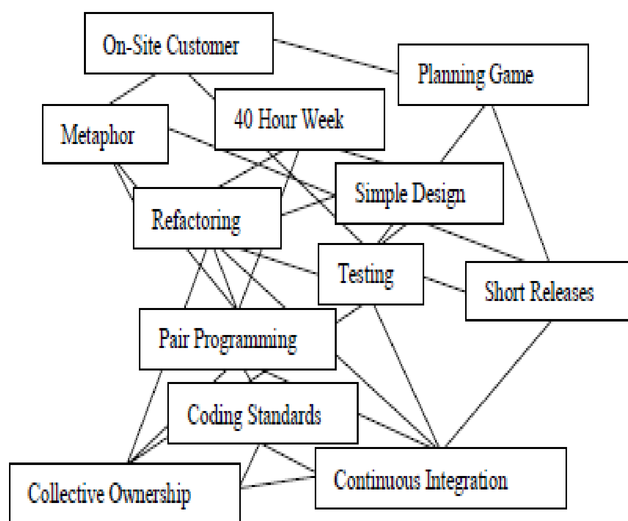

Fig1: Dependencies on Practices

XP has four basic activities, *coding*, *testing*, *listening* and *designing*, which are conducted by five major roles, *programmer, customer, tester, tracker,* and *coach.* Iteration is a key concept in XP. The time constant in the different iterations range from seconds to months, see Figure 1. The major reason for this was that most of the projects studied were small parts of larger projects. Therefore, they could not implement all the practices of XP; they did what they could in their own situation. It was not clear from the study which other XP practices they would have implemented if they had had the chance. However, since a large part what we wanted to learn was which practices work in our culture, we were interested in the empirical rather than the hypothetical, anyway. Because of the position in the schedule, as well as the fact that they were in larger projects, it was impossible to collect reliable quality and productivity metrics. However, it appeared that their productivity compared favorably to standard development methodologies in the company. It was significant to note that every person interviewed was enthusiastic about the methodology. All intended to continue their practices, and expand them where possible. The list describes the Extreme Programming Practices

Small Releases – All projects used small releases in some form. The size of the development intervals was from two to four weeks. Where the XP development was part of a larger project, the development intervals culminated in deliveries into the official code base of the project. One team punctuated each interval with a demo to themselves and others on the project. They reported that this was a strong morale booster on the team. One project's data showed that they had not made substantial improvement in the accuracy of their development estimates over several iterations. They admitted, however, that they had not re-estimated after each iteration, but plan to do so in the future.

Metaphor – No project had a metaphor. This is consistent with reports from Kent Beck , who stated that people tell him that they do XP, "..except metaphor, of course."[18]. Others have also noted that people have difficulty understanding Metaphor [19]. This is certainly one major reason that no project used metaphor. The other major reason is that XP's Metaphor intends to fill some of the purpose of traditional software architecture, namely creating a shared vision of the system to be built. Every project in the study did produce an architecture. This was no doubt partly due to the influence of existing practices, but no project even considered creating a metaphor rather than an architecture. Nobody was interested in metaphor.

Simple design – Projects did not highlight simple design as an important part of their XP process. This should not be construed to imply that they did not have simple designs, but rather that it was not an important difference from their traditional processes.

Testing – All projects intended to require that tests be submitted along with code, creating a body of automated regression tests. One project followed this rigorously. Other projects followed it partially. The major reason for this was schedule pressure with focus on delivered functionality. A related reason was the time and effort were not available to set up an automated regression testing system, particularly where the XP project was part of a larger project. All projects were for software to be sold to multiple customers, so it was not realistic to have a customer write and execute acceptance tests. However, every project did have an extensive system verification program. In this model, the system testers functioned as "surrogate customers", writing and executing acceptance tests.

Refactoring – Refactoring did not figure prominently in the projects. The only project to refactor frequently was a forward looking work project with three people on the team. The other projects indicated that they were not opposed to refactoring, but there really hadn't been a need to do so. This may be a reflection of more up-front design than is typically done in an XP project.

Pair programming – All projects did some form of pair programming, but each did it a little differently. No project required it for every line of code; in fact in every project, pair programming was voluntary. In one project, developers began by doing all programming as pairs, but found it was too inefficient. So they programmed the simple code individually, but paired up on the difficult code. Another project had two developers, located 2000 miles apart. They tended to write code individually, but debugged their code together, using a shared desktop. One developer pointed out that this was actually more convenient, because they weren't crowded against each other. Code inspections are standard practice in Avaya. In one instance, pair programming was allowed to replace code inspections. The project did not have data to indicate whether one was more effective than the other in finding errors.

Collective ownership – Code ownership practices varied from project to project, due in part to constraints of the surrounding projects. Where collective ownership was practiced, there was a practice of de facto code ownership:

people gained natural expertise in certain parts of the system, and made the bulk of changes there. One project codified this practice into a "lightweight ownership" policy: one could change any of the code, but needed to check with the owner of the code for advice.

Continuous integration – In most cases, projects worked within the larger project methodology of integration. This was generally weekly integrations into the main software base, although the XP sub-projects were able to integrate more often. In one standalone case, the team members integrated continuously.

On-site customer – No project had an on-site customer. As stated above, this model is not practical where one has many customers or potential customers. In addition, it is usually not desirable; the project gets only a single view among many customers. Projects continued to use a surrogate customer model, where an aggregate view of customer needs is created.

Coding standards – Avaya has had a tradition of coding standards. The XP projects followed their pre-existing coding standards.

IV. Analysis of Rational Unified Process

We begin with the history of the methodologies, and then move towards the underlying philosophies and the project types, for which the methodologies are suitable. RUP is created by the well-known triple, Jacobson- Booch-Rumbaugh, launched in its first version 1998. Jacobson began the development of the use-case based approach at Ericsson in the 1980's. RUP has evolved in conjunction with the Unified Modeling Language (UML) [8]. RUP is based on the originators' and others practical experience from software engineering, and has evolved further during the years, as well as the UML language. RUP is designed for large product development projects. Even though books are published on the methodology, the main distribution channel is though purchasing of licenses for the tool support for the RUP methodology, offered by Rational Software, which now is owned by IBM. XP has its origins in practical applications in projects during the 1990's. Beck and Cunningham have packaged their experiences into XP, originally from a project at Chrysler. It is a lightweight method for small to medium sized software development teams. XP is intended to meet the demands of a context with unclear and volatile requirements. There is nothing commercial in the methodology; instead there is a set of people – a community – who evolve and develop tool support (freeware and shareware) to support XP development projects. The origin of RUP and XP are similar. They are both based on experience from software engineering. Both are evolved during the same decade, although RUP has its roots earlier. There are two different underlying philosophies behind RUP and XP. RUP takes to a large extent a technical management perspective while XP focuses on the development staff. RUP is designed to support large projects, while XP is originally designed for small to medium sized projects, for which type of projects several experience reports are published, see e.g. [9][13][19]. The distribution of the methodologies is different; RUP is commercial and XP is freeware. On the *technical* side, RUP provides the organization a large package of development tools and documents. It is delivered online via the web, and updated in new releases. It can be tailored and extended to suit the individual organization's needs. One major sales argument for RUP is the integrated tool-suite. XP on the other hand strives towards simplicity. It is not connected to specific tools but lets the user choose which tools to use. Tools are developed in the XP community, which support specific practices, e.g. *Junit* for the testing practice. RUP is a large collection of processes, artifacts and roles. This must be scaled down for most projects except for the very largest ones. XP starts in the other direction, with a minimal core of values and practices, which has to be scaled up to fit larger contexts. The *financial* issues are different in the distribution and support of the methodologies, since RUP is a commercial product and XP is freeware. The financial power behind RUP is used for marketing giving more visibility to RUP. Rational Software is owned by IBM, which has good reputation in the software industry. RUP provides continuous updates, which enables the users being up to date regarding development methodologies. On the other hand, why should one pay for something that can be achieved for free? Effort must be spent on tailoring RUP, why should an organization then pay for it as well? [10] XP offers the freeware solution, which is financially advantageous, but may cause social reactions. The *social* aspects of RUP and XP are also related to the commercial versus freeware discussion. Larger software development companies are used to buying software licenses, and hence buying licenses for methodology is quite natural. The freeware principle behind XP is met with skepticism. Can something that is for free be good? The situation is very much like the open source situation. Free software is offered from the open source community and software is licensed from commercial companies, e.g. the Linux operating system versus Microsoft Windows. The choice is of course primarily technical and financial, but there is a significant social aspect. Smaller organizations and technical staff show a tendency to be more in favor of the freeware/open source approach, while large organization and management are in favor of the license approach. The good reputation and financial strength behind RUP are management arguments, while on the technical level, people know that both approaches need tailoring and hard work – hence they choose the method which is cheapest, least complex, and puts the technical work in focus. The RUP process as such is guided by a tool, and there are suitable tools for e.g. modeling that interface smoothly with the methodology. As the methodology is so extensive, this is absolutely necessary, to guide the user. This is also a part of the commercial success of RUP. XP does not proclaim any specific tools. There are tools offered by the community, e.g. *Junit*, but any kind of CASE tools and project management tools can be used in XP. However, it is worth noticing, that in its original form, whiteboards, paper cards and pens are the most mentioned tools in XP. What characterizes the developers and organizations using RUP and XP respectively? XP focuses on the individual developer,

empowering the technical level in the organization. It is based on direct communication between stakeholders, and requires courage, as openness and honesty are important. This requires the staff and organizations acknowledge and maintain these kinds of characteristics and values. It requires team workers solving problems in teams, and not feeling discomfort for peer reviews. RUP does not focus on the individual developer, but emphasizes the roles, which are tailored to specific projects. It prescribes documentation, which puts demands on the staff to be motivated to spend effort on preparing and maintaining the artifacts. The origin of the methods are different, RUP originates from large projects and organizations, and XP from the small. This fact permeates the methodologies as such, as well as its advocates and critics. RUP is a top-down methodology, advocated by management while XP is a bottom-up methodology, advocated by the technical staff.

V. Conclusions

In this paper, we have found the similarities and differences between Rational Unified Process and Extreme Programming methodologies. Although many keywords and key values are the same, the two methodologies are quite different. Common values are user involvement, iterations, continuous testing and flexibility. The implementation of these values are however very different. Rational Unified Process offers an extensive process description, comprising arte-facts, roles, activities, integrated tool-suites etc. XP on the contrary stresses values and principles, rather than prescriptive instructions, and focuses freedom and simplicity. The distribution channels are different, Rational Unified Process being a commercial product by a large company, and XP is freeware, maintained by a community of volunteers. We conclude from this analysis that the two in many aspects are in contrast. The situation is very similar to the Windows vs. Linux case. One is commercial, the other is freeware. One tends to be advocated by managers, the other by engineers. Still both are operating systems for personal computers. It is important to be aware of this social aspect in the selection of RUP. Which of the two is best suited for certain types of projects needs to be further investigated in empirical studies.

References

[1] K. Beck, Extreme Programming Explained, Reading, MA: Addison-Wesley, 2000.
[2] M. Beedle, et al., "SCRUM: A Pattern Language for Hyperproductive Software Development," in Pattern Languages of Program Design 4, N. B. Harrison, B. Foote, and H. Rohnert, eds., Reading, MA: Addison-Wesley, 2000, pp.637-652.
[3] J. O. Coplien, "A Generative Development-Process Pattern Language," in Pattern Languages of Program Design, J. O. Coplien and D. Schmidt eds., Reading MA: Addison-Wesley, 1995, pp. 183-238.
[4] W. Cunningham, "EPISODES: A Pattern Language of Competitive Development," in Pattern Languages of Program Design 2, J. M. Vlissides, J. O. Coplien, and N. L. Kerth, eds., Reading, MA: Addison-Wesley, Reading MA 1996, pp. 371-388.
[5] K. Beck and M. Fowler, Planning Extreme Programming, Reading, MA: Addison-Wesley, 2000.
[6] R. Jeffries, Extreme Programming Installed, Reading, MA: Addison-Wesley, 2000.
[7] J. W. Newkirk and R. C. Martin, Extreme Programming in Practice, Reading, MA: Addison-Wesley, 2001.
[8] K. Beck et al, "The Agile Alliance Manifesto," available http://www.agilemanifesto.org/principles.html.
[9] A. Cockburn, Agile Software Development, Reading, MA: Addison-Wesley, 2002.
[10] M. Fowler and J. Highsmith, "The Agile Manifesto," Software Development, vol.9, no. 8, pp. 28-32, Aug. 2001.
[11] IBM (Smith, J.), *A Comparison of the IBM Rational Unified Process and eXtreme Programming*, http://www3.software.ibm.com/ibmdl/pub/software/raional/web/whitepapers/2003/TP167.pdf
[12] Jacobson, I., Booch G. and Rumbaugh, J, *The Unified Software Development Process*, Addison-Wesley,1999.
[13] Karlström, D., "Introducing Extreme Programming - An Experience Report", *Proceedings Third International Conference on eXtreme Programming and Agile Processes in Software Engineering*, 2002.
[14] Karlström, D. and Runeson, P., "Integrating Agile Software Development into Stage-Gate Managed Product Development", technical report CODEN : LUTEDX (TETS-7203) / 1-34 / (2004) & local 16, 2004.
[15] Kitchenham, B., Linkman, S., and Linkman, S., "Evaluating Novel Software Engineering Tools", *Proceedings The 7th International Conference on Empirical Assessment in Software Engineering* (EASE 2003), Keele University, Staffordshire, UK, pp. 233-247, 2003.
[16] Kruchten, P., *The Rational Unified Process – An Introduction*, Addison-Wesley 2nd edition, 2000.
[17] Lindland O. I., Sindre, G. and Sølvberg, A., "Understanding in Conceptual Modeling", *IEEE Software*, March, pp. 42-48, 1994.
[18] Robson, C., *Real World Research*, Blackwell Publishers, Oxford, 2nd edition, 2002
[19] Schuh, P., "Recovery, Redemption, and Extreme Programming", *IEEE Software* December, pp. 34-41,2001.
[20] Scott, K., *The Unified Process Explained*, Pearson Education, 2001.