

Build your Project using Extreme Programming

#2 of a Series, by Pavan Kumar Gorakavi, M.S., M.B.A, G.M.C.P, C.A.P.M.

1. What is Extreme Programming?

Extreme Programming is a software development methodology that has earned its importance in the arena of agile software development methodologies. Kent Beck initially formulated the Extreme Programming approach.

Due to different disadvantages of conventional software methodologies, in the late 90s project developers were looking for alternative software development methodologies. Extreme Programming appeared. This approach resolves many of the problems caused by conventional software development methodologies. The term 'extreme' comes from taking these common principles and practices to extreme level [1].

Extreme Programming is a disciplined methodology that stresses customer satisfaction. It underscores the concept of 'deliver when needed'. Unlike conventional methods, agile practices maintain minimal re-factoring cost. Agile processes primarily focus on high interaction, working models, customer collaboration, regular feedbacks, and flexibility for changing requirements. Extreme Programming encourages a high degree of interactions between team players, including developers, testers, managers, business owners and others. The practice advises the team to possess a lucid information system. Extreme Programming emphasizes not just testing, but testing well. Test cases and tests are created at all stages of coding. As Extreme Programming provides early feedback, it will reduce failure cost. It provides a flexible environment for both customers and developers for changing requirements.

Extreme Programming needs an effective team with size ranging from 5-20 team players. Small teams handling smaller modules yield efficient output in XP practice. XP address the problem of huge development cost by providing early prototypes. This practice helps developers, testers, managers, customers and other key players to have frequent interaction. Extreme Programming can be most-efficiently applied when the team environment is highly communicative, has small flexible groups, de-fragmented modules, proper testability and productive working circumstances.

2. Life Cycle of Extreme Programming

Extreme Programming can be implemented by disciplined methodology that focuses primarily on customer satisfaction. The Life cycle of extreme programming includes Planning, Iterative development phase, Maintenance phase and a Death phase. In the Planning phase user stories are written, Iterative implementation planning is performed, efforts are estimated, and priorities of the features are decided. In the Iterative development phase, functional modules are developed using standard software development life cycle process: analysis, design, development and testing. This is iterative in nature.

'Pair Programming' is a unique feature associated with XP. The Iterative development phase includes continuous reviews, continuous integration and quick feedback. Maintenance phase includes new small releases with customer experience. Once the XP effort reaches a point where there are no remaining user stories, the product life cycle reaches a death phase. This is illustrated in figure 2, below.

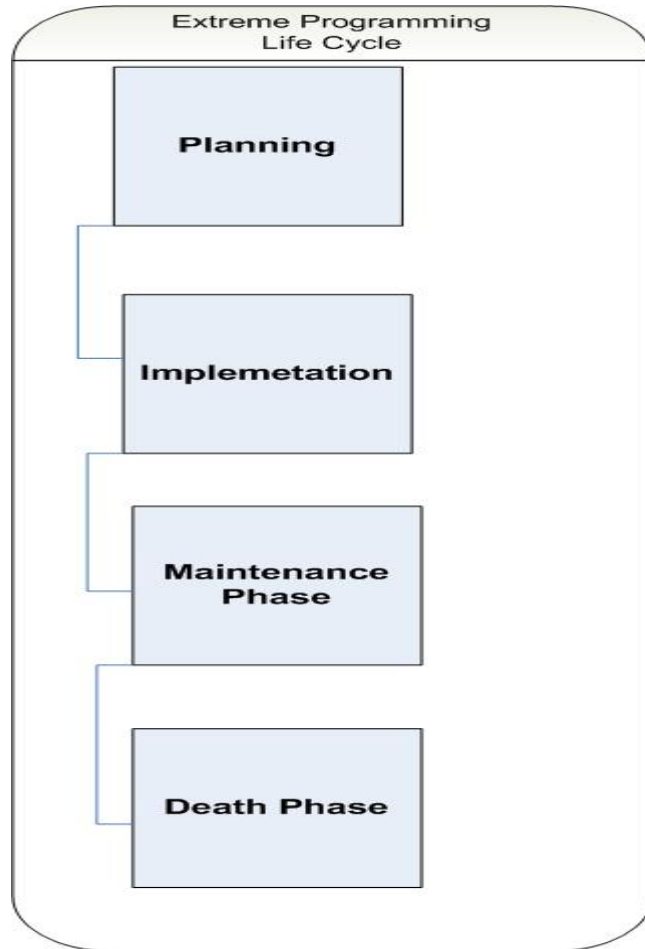


Figure 1: Life cycle of Extreme Programming

A. Planning Phase

User stories are initially written in this phase. User stories are brief: 3-4 lines of description about usage scenario. User stories briefly describe initial details to identify effort estimates, risk factors involved, and begin to identify basic acceptance test cases. User stories functions similar to use-cases but at very high level. Often user stories are compared with requirement documents; unlike requirement documents user stories outline high-level statements.[5] User stories avoid technical specifications, user interface layouts and any other micro details. Either business owners or customers generally write the user stories.

- Student registers for courses
- Student can drop any course
- Student can leave a note to advisor
- Student can purchase parking ticket
- Student can pay their fees

Figure 2: User Stories for student management system

User stories are generally written on index cards. Index cards include high level of descriptions about user stories, priority of the story, estimates, and any other details. Prioritization can generally be based on High/Medium/Low or a number index. A sample index include is depicted in figure 3, below.

CR/SR no. 1234
Student can register for a course.
Priority: Medium Estimated hours: 24

Figure 3: Index cards

Once user stories are developed, the XP team performs a formal meeting with business owners; developers, testers and other key players identify the priority of the user stories, and formally decide about a release plan. The team develops the estimates for each story and projects how much a team can produce in a given time interval. Project planning is generally performed by either time or scope of the project. Previous historical statistics can be used to identify the estimates for the subsequent iterations. Some of the following statistical measures can be used to analyze prior history during planning.

Project velocity = \sum Estimates of user stories that were finished in an iteration.

Error Tolerance = (Total burned hours)-(Total Estimated hours)

Resource utilizations rate = $\frac{\text{Total estimated hours}}{\text{Total development hours}}$

Load factor = $\frac{\text{Total work hours allocated for a given iteration}}{\text{Total development hours}}$

An iteration planning session begins the iteration. User stories scheduled for next iterations, failed acceptance test cases, and any other backlog items are considered as part of this scheduled meeting. The high-level tasks are broken down into smaller tasks with technical outlines. Developers estimate the tasks and consider historical statistics like project velocity, resource utilization rate, load factor, and error tolerance while planning estimates for the next iteration. Project velocity can be calculated both at week or day level. Players need to consider about user stories being snowplowed. Planning should accommodate refactoring: either change in task or story estimates are possible.

B. Iterative Development Phase

After the user story tasks are broken down into smaller tasks, the schedule set in the planning stage is broken down to a number of iterations that will each take one to four weeks of implementations. Infrastructure projects receive high priority in the initial iterations and the customer selects the stories for subsequent iterations. During Iterative Development, a number of approaches that are relatively unique are applied, including Pair Programming, Standup Meetings, and CRC Cards.

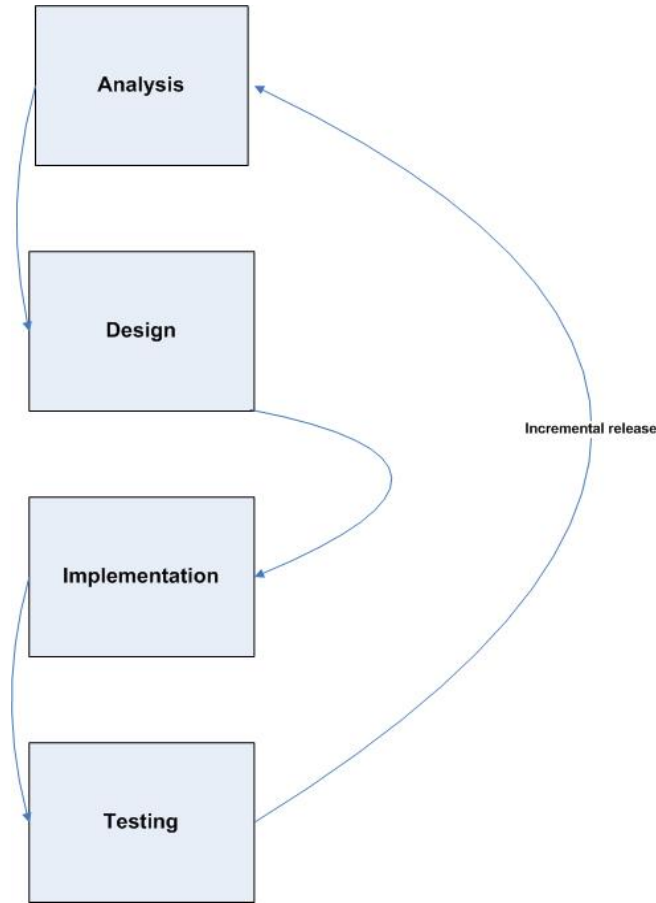


Figure 4: Iterative SDLC

Pair Programming

Pair Programming is an innovative concept of two people sitting together at a single workstation to produce a solution. One person codes the program tactically while other person thinks strategically. This approach produces the same functionality when compared with two different people sitting at two different terminals. Researchers assure a high quality product with pair programming methods. Chrysler had practiced pair-programming in developing their compensation system[3].

Pair programming encourages pair-analysis and pair-design to have smooth product delivery. Pair-analysis should include development directions and strategy outlines during the complete life cycle. As two brains works together, there is always a better chance of yielding a good solution rather than a simple solution. Pair-design helps to identify the defects in a very early stage, reducing defect tunnel vision. With the partner discussing the other developer's vision, programmers will have a better understanding of the requirement. After design is completed, the pair team starts their implementation. One of the developers will be in driver's seat and other will be the strategic controller. Unit testing will be performed after pair development by having the developers develops test cases together and executing the results.

Laurie Williams, Robert R. Kessler, Ward and Ron performed [2][4] an experiment in class producing quantitative results supporting the pair-programming results in industry. The students completed four assignments over a period of six weeks. Thirteen individuals and fourteen collaborative pairs completed each assignment. The pairs always passed more of the automated post-development test cases run by an impartial teaching assistant (see Table 1 below). (The difference in quality levels is statistically significant to $p < .01$.) Their results were also more consistent, while the individuals varied more about the mean.

	Individuals	Pair – Team
Program 1	73.4%	86.4%
Program 2	78.1%	88.6%
Program 3	70.4%	87.1%
Program 4	78.1%	94.4%

Table 1: Percentage of Test Cases Passed
Reference:[2],[4]

Advantages of Pair Programming: Errors are identified upfront. This reduces the cost of software maintenance significantly. Design reviews are part of regular pair discussions, which helps in making the design simple and reduces re-factoring cost. Pair-programming also improves cross training between developers.

Standup Meeting

Standup Meetings are encouraged in XP practice. Standup meeting reduces the disadvantages of regular long meetings, helps to make team focus on the target, and makes everyone acquaintance with overall scope of the project. Communication flow is the basic purpose of the meeting. The scope of the meeting is to communicate problems, make team focus on solutions, discuss about failed acceptance test cases, unfinished tasks, check for application dependencies and re-factor the requirements if needed.

CRC Cards

CRC cards are descriptive representations of objects. CRC cards deals with class, responsibilities and collaboration. CRC cards are used in system design. CRC cards are designed using object technology. CRC represent an object with responsibilities described towards left and collaborations towards right. Though there is a comment that CRC way of representations lacks conventional design document, graphical representation makes design obvious. See the sample CRC card, illustrated below.

Student	
Register for a course	Registration
Pay fees	Authentication
Cancel a course	
Login	
Look for Grades	

Figure 5: CRC Cards

C. Maintenance Phase

After the initial release when all acceptance test cases are completed, any subsequent iteration will fall under maintenance phase. In maintenance phase, developers need to handle the request for current production system and next iteration tasks. This development effects development velocity.

D. Death Phase

When customers or business clients are done with the user stories and there are no more new stories, then we consider the XP cycle to be in death phase.

3. Different Actors While Practicing XP

There are different players playing active role while implementing Extreme Programming. Some of them are Programmers, Customers, testers, trackers, coaches, customers, business owners, consultant and managers.

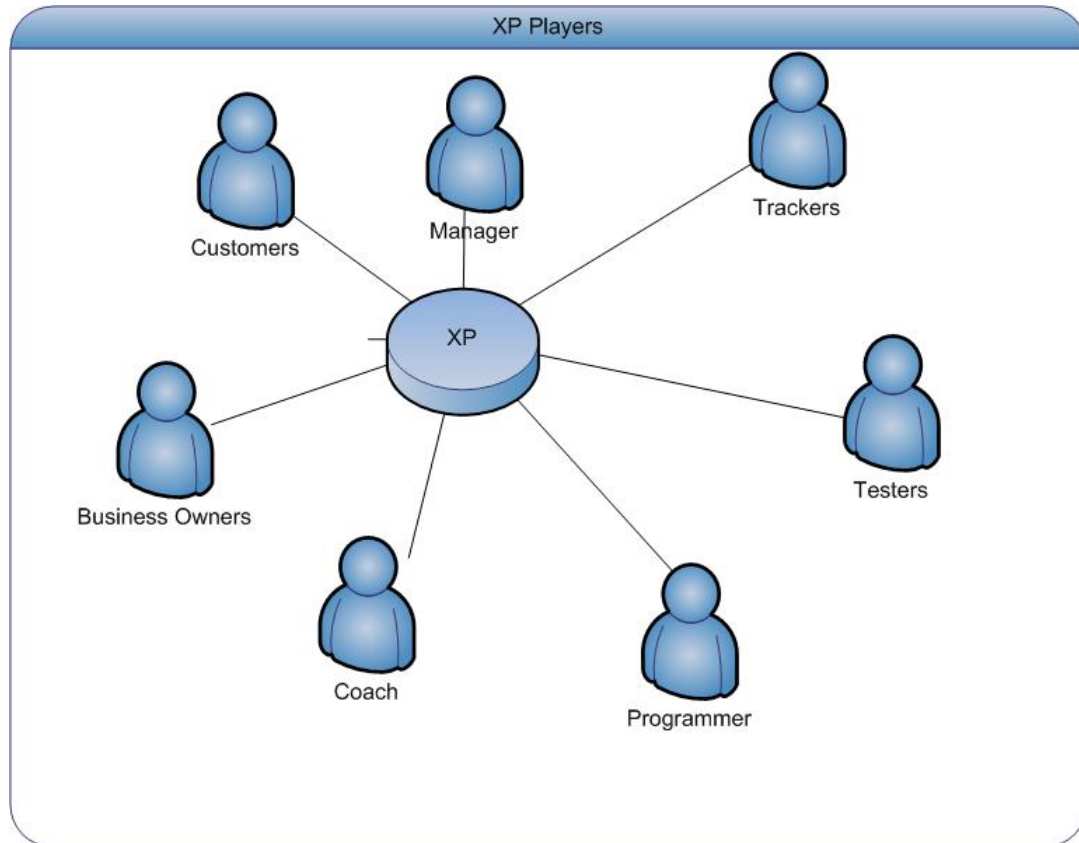


Figure 6: Key Stakeholders

Conclusion

Extreme Programming is an efficient methodology that can yield good results when good planning, small releases, simple design, efficient testing and collective ownership is practiced. As stated by Beck, the XP methodology is by no means suitable everywhere, nor have all its limits identified. Extreme Programming is generally aimed for small and medium sized teams.

References

- [1] Beck. K. Extreme Programming explained: Embrace change, Addison- Wesley.
- [2] Laurie Williams and Robert R. Kessler, Strengthening the Case for Pair-Programming <http://www.cs.utah.edu/~lwilliam/Papers/ieeeSoftware.PDF>.
- [3] A. Anderson, Beattie, Ralph, Beck, Kent et al., "Chrysler Goes to "Extremes", " *Distributed Computing*, vol. October 1998, 1998, pp. 24-28.
- [4] L. A. Williams and R. R. Kessler, "All I Ever Needed to Know About Pair Programming I Learned in Kindergarten," in *Communications of the ACM*, vol. 43, no. 5, 2000.
- [5] Stephens, Matt, and Doug Rosenberg. Extreme Programming Refactored: The Case Against XP. Apress.

About the Author

Pavan Kumar Gorakavi is working as a Senior Software Developer in Dallas, TX. He is settled in Dallas, TX with his family (wife Swapna Gorakavi and son Anish Gorakavi). He is VP - programs for *asapm* Young Crew. He is also acting as Associate Director [Marketing] for PMI-ISSIG. Pavan earned his Bachelor's degree in computer science from Jawaharlal Nehru Technological University and Masters in computer science from Lamar University. He did his MBA from University of Texas at Dallas and GMCP from Southern Methodist University. Pavan holds SUN, IBM and PM Institute certifications.

Pavan Gorakavi authored a book on 'Artificial Intelligence' published by Rahul publications - India, and 'Digital Electronics' published by Subhash publications, India. His research interests are Artificial Intelligence, Agile methodologies, and Software development in ADA, Prolog and Java. You can reach Pavan at gorakavi@gmail.com.

About this Series

This article is the first in a series by Mr. Gorakavi on Agile, posted on the *asapm* website; watch for the others in the series. And, although the concepts of Agile are most-common in Software Development projects, increasingly Agile and Lean PM methods are also turning up in many other project areas, including Engineering and Manufacturing, where some assert they actually originated.

