International Journal for Scientific Research (IJSR)
المجلة الدولية للبحوث العلمية
Vol. (2), No. (3)
IJSR
March 2023
الإصدار (2)، العدد (3)

A Proposed Best Practices for Agile Approach - XP

Thamer Abdel-Hamid Shoukry

Ph.D. Student, Software Engineering, Putra University, Malaysia

Rashed Aly Gaber

Ph.D. Student, Information Technology, Philadelphia University, Jordan

Abstract:

This paper is interested in proposing a set of proposed practices for eXtreme Programming (XP) approach to improve the quality of applying this approach in the domain of the software development process. This paper clarifies the basic concepts of agile methods and presents the fundamentals and features of XP approach. life cycle phases of XP approach include six phases: exploration, planning, iterations to release, production, maintenance, and death. Each XP phase can be achieved through performing a set of activities or steps. The researchers developed a set of improved steps for achieving XP phases. The researchers also propose a quality assurance approach for applying XP approach. The proposed quality assurance approach can be used for assuring the quality of achieving XP phases. Then, the deviation between the actual quality and the acceptable quality level can be identified and analyzed. The weaknesses of the software development practices can be discovered, treated to improve the quality of each phase, and avoided in further phases. The strengths of the proposed practices are utilized to increase the quality of achieving the software projects.

Keywords: Agile Methods Software Development; XP Approach; Agile Best Practices; XP Life Cycle

1- Introduction and Problem Definition

Software Development (SD) is a mentally complicated task. Therefore, different software development methodologies and quality assurance methods are used to attain high-quality, reliable, and bug-free software [17]. In recent years, agile software development methods have gained much attention in the field of software engineering [27]. A software development method is said to be agile software development method when a method is people focused, communications-oriented, flexible (ready to adapt to expected or unexpected change at any time), speedy (encourages rapid and iterative development of the product in small releases), lean (focuses on shortening timeframe and cost and on improved quality), responsive (reacts appropriately to expected and unexpected changes), and learning (focuses on improvement during and after product development) [1].

Agile software development is an iterative and incremental approach that is performed in a highly collaborative manner to produce high-quality software that meets the changing needs of its stakeholders. Agile software development methods offer a viable solution when the software to be developed has fuzzy or changing requirements, being able to cope with changing requirements throughout the life cycle of a project [7]. Agile software development methods include XP, Scrum, Crystal, Feature Driven Development (FDD), Dynamic System Development Methodology (DSDM), and Adaptive Software Development (ASD) [4].

- XP is the best-known agile method that is driven by a set of shared values including simplicity, communication, feedback, and courage. The XP values, practices, and life cycle will be explained in the next section of this paper.
- Scrum is an iterative and incremental approach for managing software projects in a changing environment. Each iteration aims to produce a potential set of software functionality.

IJSR, VSRP Publishing, UK ISSN 2755-3418 (Online)
Website: https://ijsr.vsrp.co.uk

- Crystal methodologies focus on incremental development which may be in parallel. Each increment may take several iterations to complete. The tunable project life cycle that is common for all Crystal methodologies is: envisioning, proposal, sales, setup, requirements, design and code, test, deploy, train, and alter [1]. Crystal family of methodologies provides guidelines for policy standards, tools, work project, and standards and roles to be followed in the development process.
- FDD is a model-driven and short-iteration approach for developing software. It focuses on the design and building phases. FDD provides guidelines, tasks, techniques, and five sequential processes: develop an overall model, build a feature list, plan by feature, design by feature, and build by feature [24].
- DSDM provides a framework that supports rapid, iterative, and collaborative software development for producing high-quality business information systems solutions [15]. The basic principle of DSDM is that the resources and timeframe are adjusted and then the goals and the required functionality are adjusted accordingly.
- ASD offers an agile and adaptive approach to high-speed and high-change software projects. ASD replaces the static plan-design life cycle with a dynamic speculate-collaborate-learn life cycle. ASD focuses more on results and their quality than the tasks [13].

XP is one of the most popular agile development methods. Therefore, it is the main concern of this paper. The XP process is characterized by short development cycles, incremental planning, continuous feedback, and reliance on communication and evolutionary design [27]. It is designed for use with small teams that need to develop software quickly and in an environment of rapidly changing requirements.

Although the many advantages and features of XP approach, using it for developing software doesn't guarantee the success of this process at an acceptable level of quality. In addition, software projects are faced with many challenges that may lead them to failure. Therefore, there is a need to assuring the quality of software development. Quality assurance is all the planned and systematic activities implemented within the quality system, and demonstrated as needed, to provide adequate confidence that an entity will fulfill the requirements for quality [11]. This paper focuses on proposing a set of best steps for achieving each XP phase, evaluating the quality of achieving this phase, and determining the deviation of achieving the phase to improve the quality of software development.

2- eXtreme Programming (XP) Approach

Extreme Programming was developed at Chrysler by Kent Beck while working on a payroll project as a member of a 15-person team. Beck continued to refine and improve the XP methodology after the project was completed until it gained worldwide acceptance in 2000 and 2001 [14].

The XP software development process focuses on iterative and rapid development. XP approach stresses communication and coordination among the team members at all times; and requires cooperation between the customer, management, and development team to form a supportive business culture for the successful implementation of XP [1].

It is designed for use in an environment of rapidly changing requirements. It helps to reduce the cost of change by being more flexible to changes. XP is characterized by six phases: exploration, planning, iterations to the first release, productionizing, maintenance, and death. XP is a software development discipline in the family of agile methodologies that contributes towards quality improvement using a dozen practices [17]. XP consists of twelve practices, which are planning game, small

releases, metaphor, simple design, testing, refactoring, pair programming, collective code ownership, continuous integration, 40-hour week, on-site customer, and coding standard [19]. Figure (1) illustrates the XP values, practices, and phases.

2-1 XP Values

XP is driven by a set of values including simplicity, communication, feedback, and courage.

- Communication: An Agile method emphasizes face-to-face communication within the team and with the customer who is closely involved with the development process [21]. XP requires direct communication among all members to give the developers a shared view of the system which matches the view held by the users of the system.
- Feedback: Software developers should always have a way of getting information about the development process. Feedback relates to many dimensions that include the system, customer, and team. Feedback from the system and the team aims to provide project leaders with quick indicators of the project's progress to take corrective or supportive actions. In addition, feedback from customers includes the functional and acceptance tests.

IJSR, VSRP Publishing, UK ISSN 2755-3418 (Online)
Website: https://ijsr.vsrp.co.uk

Figure (1): XP Values, Practices, and Phases.

- Simplicity: A simple design always takes less time to finish than a complex one. Therefore, XP encourages starting with the simplest solution. Extra functionality can then be added later. Extreme programmers do the simplest thing that could possibly work and leave the system in the simplest condition possible. This improves the overall speed of development while still retaining an emphasis on working software.
- Courage: Courage means that developers are prepared to make important decisions that support XP practices. Courage enables developers to feel

comfortable with refactoring their code when necessary. This means reviewing the existing system and modifying it so that future changes can be implemented more easily. In addition, courage may include removing source code that is obsolete, no matter how much effort was used to create that source code.

2-2 XP Practices (rules)

The four core values of XP are implemented with twelve core practices: Planning Game, Small Releases, Metaphor, Simple Design, Testing, Refactoring, Pair Programming, Collective Code Ownership, Continuous Integration, 40-hour Week, On-Site Customer, and Coding Standard.

1. Planning Game: At the beginning of the development process, customers, managers, and developers meet to create, estimate, and prioritize requirements for the next release. The requirements are captured on "story cards" in a language understandable by all parties. In fact, the developers estimate the effort needed for the implementation of customers' stories and the customers then decide about the scope and timing of releases. The planning game and the story cards offer the devices to perform planning on the most detailed level for very short periods of time [18].
2. Small Releases: The development is divided in a sequence of small iterations, each implementing new features separately testable by the customer [7]. XP increases the pace of the delivery of the software by having short releases of 3-4 weeks. At the end of each release, the customer reviews the software product, identify defects, and adjust future requirements. An initial version of the software is put into production after the first few iterations. The small releases help the customer to gain confidence in the progress of the project. In addition, the small releases help the customer to come up with their suggestions on the project based on real experience.

3. Metaphor: The system metaphor is the story that customers, developers, and managers can talk about how the system works [19]. The system metaphor is an effective way of getting all members of the project team to visualize the project. It should provide inspiration, suggest a vocabulary, and a basic architecture. This is the only principle not strictly required in every XP project.
4. Simple Design: The developers must focus on designing only what is needed to support the functionality being implemented. The Developers are urged to keep the design as simple as possible, say everything once and only once. A program built with XP should be a simple program that meets the current requirements. Kent Beck stated that the right design for the software at any given time is the one that runs all the tests, has no duplicated logic, states every intention important to the programmers, and has the fewest possible classes and methods [19].
5. Testing: Testing is an integral part of XP. All code must have automated unit tests and acceptance tests, and must pass all tests before it can be released [7]. The tests are written before coding. Sometimes, this practice is called “test first”. Programmers write unit tests so that their confidence in the operation of the program can become part of the program itself. For the same reason, customers write functional tests. The result is a program that becomes more and more confident over time.
6. Refactoring: Refactoring is the process of changing the code in order to improve it by removing redundancy, eliminating unused functionalities, improving code readability, reducing complexity, improving maintainability, adapting it to patterns or even trying to make the software work in an acceptable way. Refactoring throughout the entire project life cycle saves time of development and increases quality.
7. Pair Programming: Pair programming is one of the key practices of XP. It is a programming technique that requires two programmers to work together at solving a development task while sharing the monitor, the keyboard, and the

mouse. The work may include analyzing data, creating the data model, programming, etc. The advantages of pair programming are improving productivity, the quality of the solution, and job satisfaction [26]. Moreover, it reduces the time needed for task completion, it is particularly useful in complex tasks, and it is useful for training.

8. Collective Code Ownership: This practice indicates that the code is owned and shared by all developers. Everyone is able to edit it and see the changes made by others. It tends to spread knowledge of the system around the team. The code should be subjected to configuration management.
9. Continuous Integration: Developers integrate a new piece of code into the system as soon as possible it is ready. All tests are run, and they must be passed for accepting the changes in the code. Thus, XP teams integrate and build the software system multiple times per day. Continuous integration reduces development conflicts and helps to create a natural end to the development process.
10. 40-Hour Weeks: This practice indicates that the software developers should not work more than 40-hour weeks, and if there is overtime one week, the next week should not include more overtime. People perform best and most creatively if they are rested, fresh, and healthy. Therefore, requirements should be selected for iteration such that developers do not need to put in overtime.
11. On-Site Customer: A customer always works with the development team to answer questions, perform acceptance tests, and ensure that development is progressing as expected. This customer-driven software development led to a deep redefinition of the structure and features of the system [7]. It supports customer-developer communication [18].
12. Coding Standards: This practice indicates that the developers must agree on a common set of rules enforcing how the system shall be coded. This makes understanding easier and helps in producing consistent code. Coding standards

are almost unavoidable in XP, due to the continuous integration and collective ownership properties.

2-3 XP Process

XP approach can be viewed as life cycle phases that include six phases: exploration, planning, iterations to release, productionizing, maintenance, and death [1]. Each phase can be achieved through a set of activities. Figure (2) illustrates the XP life cycle [22].

1. Exploration Phase: In the exploration phase, the customers write out the story cards that they wish to be included in the first release. Each story card describes a feature to be added to the program. At the same time, the development team gets familiar with the development environment and the addressed technology [25]. The exploration phase takes between a few weeks to a few months, depending largely on how familiar the technology is to the programmers.
2. Planning Phase: In the planning phase, the customers set the priority order for the stories, and an agreement on the features of the first small release is made. The developers estimate the necessary effort and time for each story. Then the schedule of the first release is developed and approved. The planning phase takes a couple of days.
3. Iterations to Release Phase: In the iterations to release phase, the actual implementation is done. This phase includes several iterations of the systems before the first release. The schedule is broken down into several iterations that will each take one to four weeks to implement [22]. For each iteration, the customer chooses the smallest set of most valuable stories that make sense together and programmers produce the functionality. Small releases reduce the risk of misled development. XP coding always begins with the development of unit tests. After the tests are written, the code is developed and continuously integrated and tested. At the end of the iteration all functional tests should be

IJSR, VSRP Publishing, UK
Website: https://ijsr.vsrp.co.uk
ISSN 2755-3418 (Online)

running before the team can continue with the next iteration [19]. When all iterations scheduled for a release are completed the system is ready for production.

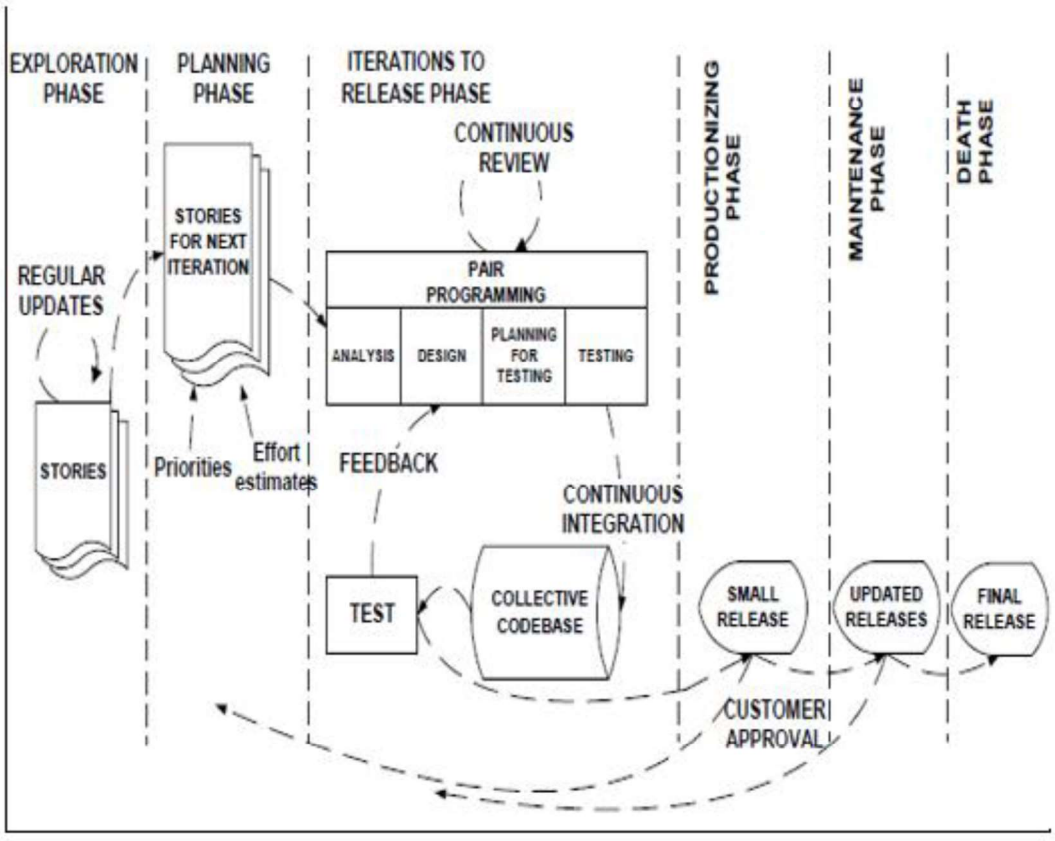

Figure (2): XP Life Cycle [22].

4. Productionizing phase: The production phase includes extra testing and checking of the functionality and performance of the system before the system can be released to the customer [22, 25]. At this phase, new changes may still

IJSR, VSRP Publishing, UK ISSN 2755-3418 (Online)
Website: https://ijsr.vsrp.co.uk

be found, and the decision has to be made if they are included in the current release. During this phase, the iterations may need to be quickened from three weeks to one week. The postponed ideas and suggestions are documented for later implementation during, e.g., the maintenance phase.

5. Maintenance Phase: After the first release is productionized, the system must be kept running in production, while remaining stories are implemented in further iterations. Therefore, the maintenance phase requires an effort for customer support tasks. Development stays in this phase until the system satisfies the customers' needs in all aspects.
6. Death Phase: Finally, development enters the death phase when the customer has no more stories to be implemented, and all the necessary documentation of the system is written as no more changes to the architecture, design, or code are made. Death may also occur if the system is not delivering the desired outcomes, or if it becomes too expensive for further development.

3- Enhanced Steps for Achieving XP Phases

In XP approach, developers communicate with each other to efficiently utilize tacit knowledge and quickly find new solutions to current challenges. Developers communicate with customer representatives to deliver the most valued features, gain rapid feedback on deliveries, and improve the customer's trust and confidence [23].

XP approach can be viewed as life cycle phases that include six phases: exploration, planning, iterations to release, productionizing, maintenance, and death [1]. Each phase can be achieved through a set of steps. The researchers propose a set of best steps for achieving each phase. In this section, the proposed best steps are presented. In these steps, if we don't tell who is responsible for performing the step, we mean that the developers and customers together must participate in doing it.

3-1 The Proposed Best Steps of "The Exploration Phase"

The XP software development process is regarded as the flow in which user stories are generated, designed, coded and unit tested, refactored, and verified. A user story is a software system requirement formulated as one or two sentences in the everyday or business language of the customers. The user stories should be written by the customers for a software project. During the development process, customers can generate new user stories and change old ones [27]. The proposed best steps required for achieving the exploration phase are:

1. Presenting and clarifying the purpose and the steps of "the exploration phase" to the customers who participate in the team.
2. Obtaining a preliminary background of the project. The background will be incremented through the next phases. The project's background includes the project's motivation, assumptions, constraints, addressed technology, and acceptance criteria.
3. Clarifying the purpose of the story cards as a tool for collecting the requirements. Each story card describes a feature to be added to the current release.
4. Presenting and clarifying the writing standards that must be considered when writing the story cards. For example, the stories must be consistent, clear, testable, and integrated with the other related stories.
5. Writing the story cards that the customers wish to be included in the current release. This step must be done by the customers.
6. Understanding the story cards. This step must be done by the developers.
7. Analyzing and validating the story cards.

3-2 The Proposed Best Steps of "the Planning Phase"

In the planning phase customers assign priorities to their stories and developers estimate the necessary effort for their implementation. Then a set of stories for the first small release is agreed upon and the release is scheduled according to the programmers' estimations [25]. If possible, near-site customers should do this with programmers in face-to-face meetings [20]. The proposed best steps required for achieving the planning phase are:

1. Presenting and clarifying the purpose and the steps of "the planning phase" to the customers who participate in the team.
2. Setting the priority order of the stories. This step must be done by the customers.
3. Identifying and negotiating the features that must be included in the current release.
4. Preparing an approved list of features needed to implement the current release.
5. Estimating the necessary effort and time for each story.
6. Preparing a proposed schedule for the current release.
7. Negotiating and approving the proposed schedule of the first release to reach to a final one.

3-3 The Proposed Best Steps of "Iterations to Release Phase"

XP promotes the concept of "small releases" [16]. The meaningful releases should be made available to users when completed. This will allow early and frequent feedback from the customers. The proposed best steps required for achieving this are:

1. Presenting and clarifying the purpose and the steps of the "iteration to release phase" to the customers who participate in the team.

2. Breaking down the schedule into several iterations. The iteration will take one to four weeks.
3. Choosing the smallest set of most valuable stories that make sense together [25] and are useful to be included in each iteration.
4. Reviewing the functionality of all iterations.
5. Selecting the iteration to be implemented. The selection process depends on the logical sequence of the current release's functionalities.
6. Developing the unit tests for the selected iteration.
7. Writing the code for the selected iteration.
8. Integrating and testing the selected iteration.
9. Ensuring that all functional tests were done before moving to the next iteration.
10. Ensuring that all iterations scheduled are completed.
11. Delivering the current release to the production phase.

3-4 The Proposed Best Steps of "Productionizing Phase"

In the productionizing phase, there are more testing and checking of the functionality and performance of the system such as system testing, load testing, and installation testing. The proposed best steps required for achieving the productionizing phase are:

1. Presenting and clarifying the purpose and the steps of "the productionizing phase" to the customers who participate in the team.
2. Performing extra testing and checking the functionality and performance of the system such as system testing, load testing, and installation testing.
3. Identifying new changes that needed to be included in the current release.
4. Implementing and testing the new changes identified in the previous step.
5. Identifying and documenting the postponed ideas and suggestions to implement them during the maintenance phase or in the next releases.
6. Delivering the current running release to the customers.

3-5 The Proposed Best Steps of "Maintenance Phase"

During the maintenance phase, the system must be kept running in production, while remaining stories are implemented in further iterations. Development stays in this phase until the system satisfies the customers' needs in all aspects [25]. The maintenance efforts can be viewed in five main activities: system maintenance, solving system crashes, end-user assistance, system enhancement, and system reengineering. The proposed best steps required for achieving the maintenance phase are:

1. Presenting and clarifying the purpose and the steps of "the maintenance phase" to the customers who participate in the team.
2. Identifying, analyzing, and documenting the circumstances that led to bugs and symptoms of the problems. Then edit programs to fix bugs.
3. Performing unit, system, and regression testing for the edited programs.
4. Identifying, analyzing, and documenting the causes of the system crash.
5. Identifying and clarifying corrective instructions that are required to prevent the system crash. These instructions may include terminating the online session, reinitializing the application, recovering lost or corrupted databases, fixing problems of local or wide networks, and/or fixing hardware problems.
6. Providing users with additional training.
7. Identifying and documenting enhancement ideas and requests.
8. Taking decisions about the enhancement ideas and requests that must be implemented in this phase or moved to the next releases.
9. Writing and testing code for the approved enhancement ideas and requests.

3-6 The Proposed Best Steps of "Death Phase"

In the death phase, the software development process has been finished. Now there is no change to architecture, design or code will be made. The proposed best steps required for achieving the death phase are:

1. Presenting and clarifying the purpose and the steps of "the maintenance phase" to the customers who participate in the team.
2. Ensuring that all predefined stories have been implemented.
3. Finalizing all project documentation.
4. Evaluating the quality of the current release and the related parts of the system.
5. Identifying and documenting the learned lessons from the project.
6. Studying the feasibility of continuing the running of the release and the system.

4- The Proposed Approach for Improving the Quality of Applying XP Approach

Applying XP approach to the software development process doesn't guarantee the success of this process at an acceptable level of quality. In addition, software projects are faced with many challenges that may lead them to failure. Therefore, there is a need to assuring the quality of software development. Quality assurance is all the planned and systematic activities implemented within the quality system, and demonstrated as needed, to provide adequate confidence that an entity will fulfill the requirements for quality [11].

The researchers propose a quality assurance approach for applying XP approach. The proposed quality assurance approach can be used to assuring the quality of achieving XP phases. Figure (3) illustrates the proposed quality assurance model. The proposed quality assurance approach includes the following activities:

1. Achieving XP phase using the proposed best steps.
2. Evaluating the quality of the achieved phase.
3. Identifying the deviation between the actual quality and the acceptable quality level.
4. Analyzing the deviation to take corrective or supportive actions.

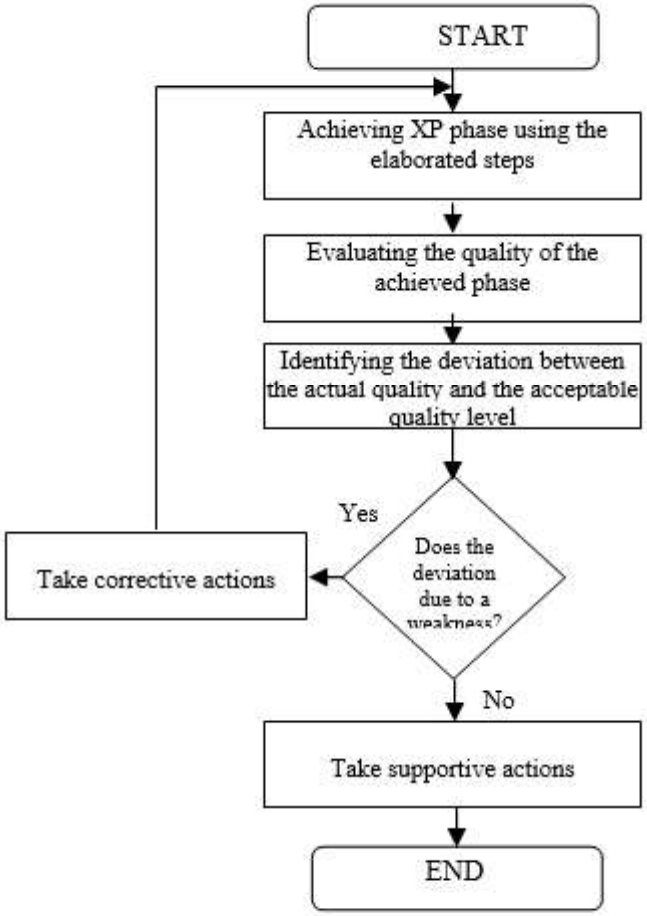

Figure (3): The Proposed Quality Assurance Approach.

Firstly, the developers must recall, present, and clarify the proposed best steps of the current XP phase to the customers who participated in the XP team. Then, the developers and customers begin to achieve the current XP phase using the proposed best steps. The proposed best steps of each phase are not having the same level of importance. Each step may have one of the cases: high importance, average importance, or low importance. Secondly, the quality of the achieved phase must be evaluated using common statistical techniques for measuring the quality. Thirdly, the deviation between the actual quality and the acceptable quality level must be identified. The acceptable quality level differs from one project to another depending on the project field and the acceptance criteria of customers. Fourthly, corrective actions must be done to the current phase if the deviation is due to a weakness in the performance. Otherwise, supportive actions may be needed for the next phases.

5- Conclusion

The main objective of this paper was to improve the quality of applying XP approach. Therefore, the researchers propose a set of best steps for achieving each XP phase and a quality assurance approach for applying the XP approach. The developers and customers can use the proposed best steps as a guiding tool for achieving each XP phase. The proposed quality assurance approach can be used to assuring the quality of achieving each XP phase. Then, the deviation between the actual quality and the acceptable quality level can be identified and analyzed.

We conclude that quality assurance practices play a very important role in increasing the probability of software development success. Applying the XP approach for developing software doesn't guarantee the success of this process. Therefore, there is a need for complementary quality assurance practices.

6- Future Work

There are many efforts that can be done in the field of XP approach in the future. Briefly, the following points are expected to be focused on:

- Proposing an approach for evaluating the quality of XP phases.
- Building a software tool for managing XP projects.
- Using XP approach to achieve higher Capability Maturity Model Integration (CMMI) levels for IT companies.
- Enhancing the calculation of software metrics related to XP projects.

References

[1] A. Qumer and B. Henderson-Sellers, "An Evaluation of the Degree of Agility in Six Agile Methods and its Applicability for Method Engineering", Information and Software Technology Vol. 50 Issue 4, 2008, pages 280–295, 2008.

[2] Alan S. Koch, "Agile Software Development - Evaluating the Methods for Your Organization", Artech House INC., 2005.

[3] Beck, K. and Andres, C., "Extreme Programming Explained: Embrace Change", Addison-Wesley, 2005.

[4] Dean Liffingwell, "Scaling Software Agility – Best Practices for Large Enterprises", The Agile Software Development Series, Pearson Education Inc., 2007.

[5] G. Gordon Schulmeyer, "Handbook of Software Quality Assurance", 4th edition, Artech House Inc., 2008.

[6] Gary Chin, "Agile Project Management: How to Succeed in the Face of Changing Project Requirements", AMACOM, 2004.

[7] Giulio Concas, Marco Di Francesco, Michele Marchesi, Roberta Quaresima, and Sandro Pinna, "An Agile Development Process and Its Assessment Using Quantitative Object-Oriented Metrics", 9th International Conference, XP 2008, Limerick, Ireland, Proceedings, June 2008.

[8] Hamid Mcheick, "Improving and Survey of Extreme Programming Agile Methodology", International Journal of Advanced Computing (IJAC), Vol. 3, Issue 3, July 2011.

[9] Helen Sharp and Hugh Robinson, "Collaboration and co-ordination in mature eXtreme programming teams", International Journal of Human-Computer Studies 66 pages 506–518, 2008.

[10] Hulkko, H. and Abrahamsson, P., "A Multiple Case Study on the Impact of Pair Programming on Product Quality", Proceedings Of ICSE, pp. 495–504, 2005.

[11] Ince, Darrel, "Software Quality Assurance - a Student Introduction", McGraw-hill international (UK) limited, 1995.

[12] Ioannis G. Stamelos and Panagiotis Sfetsos, "Agile Software Development Quality Assurance", Information science reference, Idea Group Inc., 2007.

[13] James A. Highsmith, "Adaptive Software Development: A Collaborative Approach to Managing Complex Systems", Dorset House Publishing, New York, 2000.

[14] Jeffrey A. Livermore, "Factors that Significantly Impact the Implementation of an Agile Software Development Methodology", Journal of Software, Vol. 3, No. 4, APRIL 2008.

[15] Jennifer Stapleton, "DSDM: The Method in Practice", Addison-Wesley, 1997.

[16] John Hunt, "Agile Software Construction", Springer, 2006.

[17] K.Usha, N.Poonguzhali, and E.Kavitha, "A Quantitative Approach for Evaluating the Effectiveness of Refactoring in Software Development Process", International Conference on Methods and Models in Computer Science, Delhi, India, Dec. 2009.

[18] Karlheinz Kautz and Sabine Zumpe, "Just Enough Structure at the Edge of Chaos: Agile Information System Development in Practice", 9th International Conference, XP 2008, Limerick, Ireland, Proceedings, June 2008.

[19] Kent Beck, "Extreme Programming Explained: Embrace Change", Addison Wesley, 1999.

[20] N. Wallace, P. Bailey, and N. Ashworth, "Managing XP with Multiple or Remote Customers", Third International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP2002), 2002.

[21] Noura Abbas, Andrew M. Gravell, and Gary B. Wills, "Historical Roots of Agile Methods: Where Did Agile Thinking Come From?", 9th International Conference, XP 2008, Limerick, Ireland, Proceedings, June 2008.

[22] Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, and Juhani Warsta, "Agile Software Development Methods – Review and Analysis", VTT, 2002.

[23] R. C. Martin, "Extreme Programming - Development Through Dialog", IEEE Software, pp. 12–13, 2000.

[24] S.R. Palmer and J.M. Felsing, "A Practical Guide to Feature-Driven Development", Prentice-Hall Inc, 2002.

[25] Tobias Hildenbrand, Michael Geisser, Thomas Kude, Denis Bruch, and Thomas Acker, "Agile Methodologies for Distributed Collaborative Development of Enterprise Applications", International Conference on Complex, Intelligent and Software Intensive Systems, 2008.

[26] Williams, L., Kessler, R., Cunningham, W., and Jeffries, R, "Strengthening the Case for Pair Programming", IEEE Software 17, 19–25, 2000.

[27] Yang Yong and Bosheng Zhou, "Evaluating Extreme Programming Effect through System Dynamics Modeling", International Conference on Computational Intelligence and Software Engineering (CiSE), Wuhan, China, Dec. 2009.