AGILE METHODS AND QUALITY _ A SURVEY

BODJE N'Kauh Nathan-Regis[1] and Dr G.M Nasira[2]

[1]PhD Candidate, Computer Science, Christ University Bangalore
bodje.nathan@yahoo.fr
[2]Assistant Professor, Chikkanna Govt. Arts College, Tirupur -2, India
nasiragm99@yahoo.com

ABSTRACT

Agile software processes, such as extreme programming (XP), Scrum, Lean, etc., rely on best practices that are considered to improve software development quality. It can be said that best practices aim to induce software quality assurance (SQA) into the project at hand. Some researchers of agile methods claim that because of the very nature of such methods, quality in agile software projects should be a natural outcome of the applied method.

As a consequence, agile quality is expected to be more or less embedded in the agile software processes. Many reports support and evangelize the advantages of agile methods with respect to quality assurance, Is it so ?

An ambitious goal of this paper is to present work done to understand how quality is or should be handled. This paper as all survey papers attempt to summarize and organizes research results in the field of software engineering, precisely for the topic of agile methods related to software quality.

KEYWORDS

Agile methods, quality assurance, software life cycle, software metrics

1. INTRODUCTION

Precisely from the quality point of view; the discussion at agile methods is wide and entails many facets including: differences and similarities between the traditional quality assurance procedures and Agile Software Quality Assurance, identification and evaluation of quality metrics in agile software development.

The paper will attempt to report the state of the art regarding quality achievements in agile methods and investigate on how practices and tools affect the quality in agile software development. There is a literature gap in providing a critical view of agile quality, this areas need improvement. The papers studies for this survey are from Ioannis G. Stamelos and Panagiotis Sfetsos book in agile software development quality assurance [1].

This survey paper is made up of 4 sections; where each section will analyze the paper most relevant in our senses for categorize the work done in the field. The first one will give an overview of agile and Quality. The second will describe quality within agile development. The next one will describe work done for quality within agile process management and the last chapter will show example of field experience.

2. AGILE AND QUALITY

Agile Software Methods: State-of-the-Art[2], let us know more about the concept of agile methods and quality aspect in software development. The paper gives a comprehensive analysis of agile methodologies from the perspective of software quality assurance. About agile with have a definition from three perspectives: theoretical, functional and contextualized. And the paper makes a link to relate agile and quality starting from a brief review of the traditional understandings of quality assurance to the perspective under agile methods.

2.1. Agile Definition

The theoretical definition of agile is given by Lindvall et al.(2002) as a group of software development processes that are iterative, incremental, self organizing, and emergent.

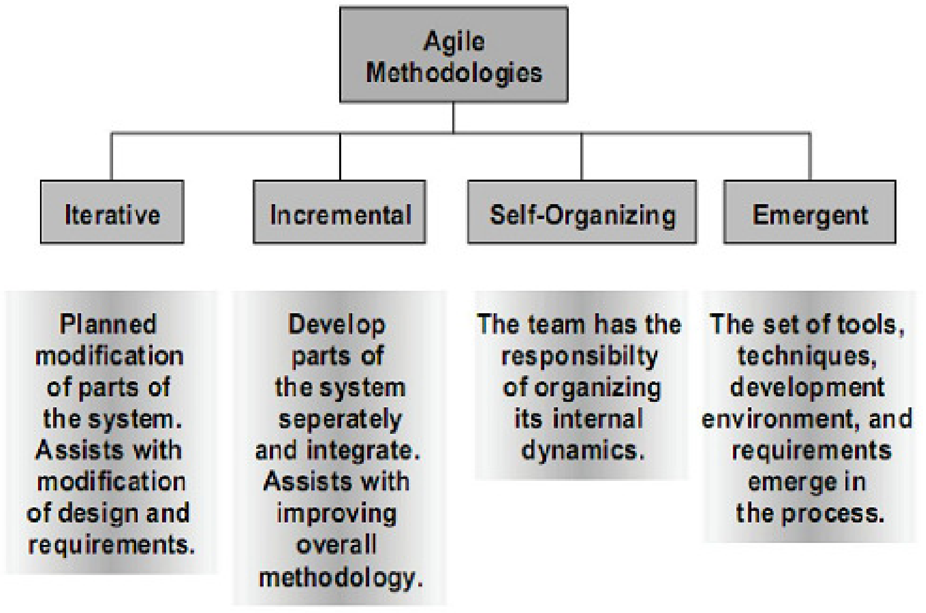

Agile Methodologies
Iterative
Incremental
Self-Organizing
Emergent
Planned modification of parts of the system. Assists with modification of design and requirements.
Develop parts of the system seperately and integrate. Assists with improving overall methodology.
The team has the responsibilty of organizing its internal dynamics.
The set of tools, techniques, development environment, and requirements emerge in the process.

Figure 1. Definition of agility - Copyright Ernert MnKandla PhD thesis University of the Witwatersrand

The functional definition according to Abrahamson et al (2002), the term “agile” carries with it connotations of flexibility, nimbleness, readiness for motion, activity, dexterity in motion and adjustability. Nonetheless, Beck (1999) defines agile methodologies as lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop software.

The contextual definition derived from what agility means in terms of certain specific software engineering concepts. Because the book was specially focus on agile software quality assurance the contextual definition of agile will be in relation with quality assurance.

2.2. Agile software quality assurance

According to Juran, quality means two things: " (1) quality consists of those product features that meet the needs of customers and thereby provide product satisfaction and (2) consist of freedom from deficiencies" (Juran and Gryna, 1988).

Meyer (2000) defines software qualiy according to an adapted number of quality parameters as defined by McCall (1977), which are correctness, compatibility, robustness, extendibility, reusability, efficiency, partability, integrity, verifiability and ease to use.

From Agile perspective, quality as McBreen (2000) said: it is the development of software that can respond to change. We can notice that quality is a rather abstract concept that is difficult to define but where it exists, it can be recognized.

The table below gives some parameters that define agile quality for extreme programming.

TECHNIQUE	DESCRIPTION
Refactoring	Make small changes to code, code behavior must not be affected, resulting code is of higher quality (amber, 2005)
Test-driven development	Create a test, run the test, make changes until test passé (amber, 2005)
Continuous integration	Quality assurance test done on a finished system, usually involves the users, sponsors, customer, etc. (huo, verner, zhu, & babar , 2004)
Pair programming	Done on a daily basis after developing a number of user stories. implemented requirements are integrated and tested to verify them. this is an important quality feature.
Face-to-face communication	Two developers work together in turn on one pc, bugs are identified as they occur, hence the product is of a higher quality (huo et al., 2004)
On-site customer	Preferred way of exchanging information about a project as opposed to use of telephone, email, etc. implementated in form of daily stand-up meetings of not more than twenty minutes (huo et al, 2004). this is similar to the daily scrum in the scrum method. it brings accountability to the work in progress, which vital for quality assurance.
Frequent customer feedback	Each time there is a release the customer gives feedback on the system, and result is to improve the system to be more relevant to needs of the customer (huo et al., 2004). quality is in fact meeting customer requirements
System metaphor	Simple story of how the system work (huo et al., 2004), simplifies the discussion about the system between customer/ stakeholder/ user and the developer into a non-technical format. simplicity is key to quality.

Table 1. Agile quality techniques as applied in extreme programming

2.3. In Resume

In this paper, an overview of agile methodologies was presented. Authors arrived with approach definitions which are philosophical and practical about agile methodology. The authors said the future trends of agile software development is to embedded innovative thinking for higher level of maturity and quality assurance; as agile process begin to enter grounds such as enterprise architecture, patterns, etc.

3. QUALITY WITHIN AGILE DEVELOPMENT

Our second paper in study is: Handling of Software Quality Defects in Agile Software Development; produce by Jörg Rech[3]. It will help us know how researcher understands quality within Agile Development. The paper show refactoring; which is an important phase for continuous improvement will add value on the quality assurance aspect built in Agile development process. The work describes a process for recurring and sustainable discovery, handling, and treatment of quality defects found in source code. In agile software development, organizations use quality assurance activities likes refactoring to tackle defects that reduce software quality.

This research was concerned with the development of techniques for discovery of quality defects. The technique used as a quality driven and experience based method for the refactoring of large scale software system.

3.1. Quality Defect Discovery

The techniques for the discovery of quality defects are based upon several research fields such as : Software inspection, code inspection, software testing and debugging, etc. Software inspection and code inspection are concerned with the process of manually inspecting software products in order to find potential ambiguities as well functional and non-functional problems (Brykczynski, 1999).

Software testing and debugging is concerned with the discovery of defects regarding the functionality and reliability as defined in a specification or unit test case Software product metrics are used in software analysis to measure the complexity, cohesion, coupling or other characteristics of the software product.

There are several more techniques and tools for quality defect discovery. They are an example of tools like : Checkstyle, FindBugs, Hammurapi or PMD.

3.2. Quality Defects on the code level

Various forms of quality defects exist. Some target problems in methods an classes, while others describe problems on the architecture or even process level. The Author choose to focus on the code level. The representatives on the code level are :

Code Smells: Abbrevation of "bad smells in code". It was described in Beck et al. (1999). Code smells are indicators for refractoring and typically include a set of alternative refactoring. Today's code smells are semi-formally described, they are at least 38 knows.

Architecture Smells: describe problems on the design level. Roock et al (2005) define 31 architetural smells wich are apply on design level but also on the code level.

Design Patterns and Anti-patterns : Gamma et al. (1994) about design patterns and Brown et al (1998) for anti-patterns represent typical patterns of good and bad software architecutre. Patterns state and emphasize a single solution to multiple problems, anti-patterns emphasize a single problem to multiple solution.

Bug Patterns : Typically found in debugging and testing activities. Allen (2002) described 15 bug patterns.

Design characteristics : Whitemire (1997) describes nine distinct and mesurable characteristics of an object-oriented design. These characteristic was focus on the similarity of two or more classes or domain level in terms of their structure, fonction,behaior or purpose. Riel, Roock et al work design heuristics wich provide support on how to construct software systems. 61 design heuristics was describe in Riel(1996) and 14 principles in Roock(2005).

3.3. Refactoring

Beside the development of software systems, the effort for software evolution and maintenance is estimated to amount to 50% to 80% of the overall development cost (Verhoef, 2000). Reworking parts of the software in order to improve its structure and quality (e.g., maintainability, reliability, usability, etc.), but not its functionality is one step in the evolution and development of software systems. This process of improving the internal quality of object-oriented software systems in agile software development is called refactoring (Fowler, 1999).

From the agile world, in general, refactoring (Fowler, 1999; Mens et al., 2004) is necessary to remove quality defects that are introduced by quick and often unsystematic development. Organizations use techniques like refactoring to tackle quality defects (i.e., bad smells in code (Beck & Fowler, 1999), architecture smells (Roock et al., 2005), anti-patterns (Brown et al., 1998), design faws (Riel, 1996; Whitmire, 1997), and software anomalies (IEEE-1044, 1995), etc.) that reduce software quality.

During the last few years, refactoring has become an important part in agile processes for improving the structure of software systems between development cycles. Refactoring is able to reduce the cost, effort, and time-to-market of software systems. Especially in agile software development, methods as well as tools to support refactoring are becoming more and more important (Mens, Demeyer, Du Bois, Stenten, & Van Gorp, 2003).

Refactoring does not stop after discovery; the refactoring used has to be documented in order to support maintainers and reengineers in later phases. These information can be stored in configuration management systems (e.g., CVS, SourceSafe), code reuse repositories (e.g., ReDiscovery, InQuisiX), or defect management systems.

3.4. Quality Defect Ontology

In software engineering, in specially in the case of quality defects for a software product, the context of defect can be describes as problems on different levels of complexity and might occur in parralel in one situation (ie, in one code fragment). Here de figure below resume the conceptual model of the quality defect ontology in the software product level.

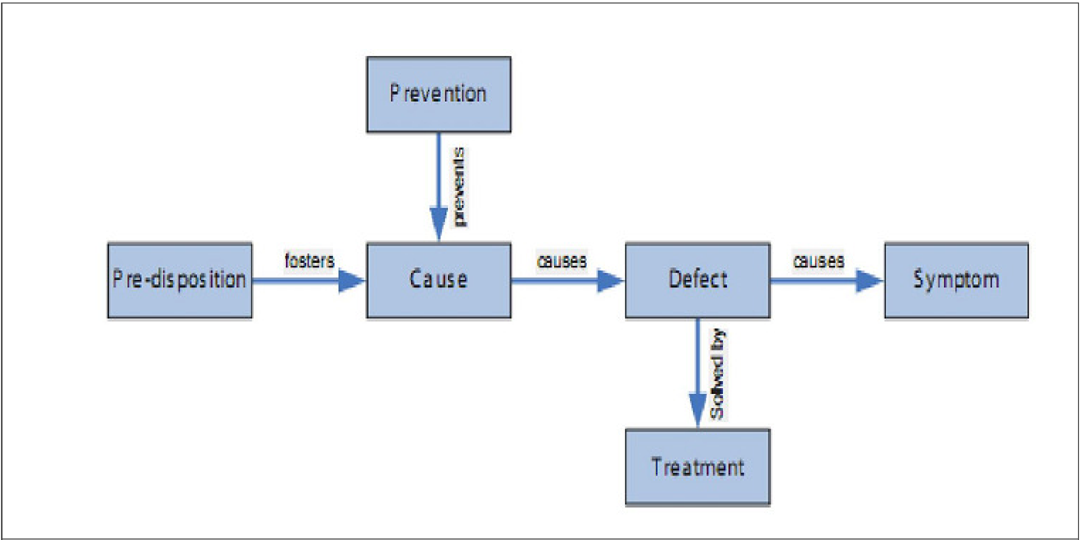


Figure 2. Conceptual model of quality defect ontology (Software Product Level)

A software system might have predispositions that foster or enable the creation of quality defects. These defects themselves have causes that are responsible for the defects being integrated into the system. The quality defects might have a negative as well as a positive effect on specifc qualities and are perceivable via specifc symptoms. Finally, the defects are solved or removed via specifc treatments after they are discovered, or the causes might be prevented by special preventive measures.

3.5. Handling of quality defects

Several quality defects are introduced into the software system and are discovered especially in the refactoring phase. The figure below show a model for defect discovery and hanling process.

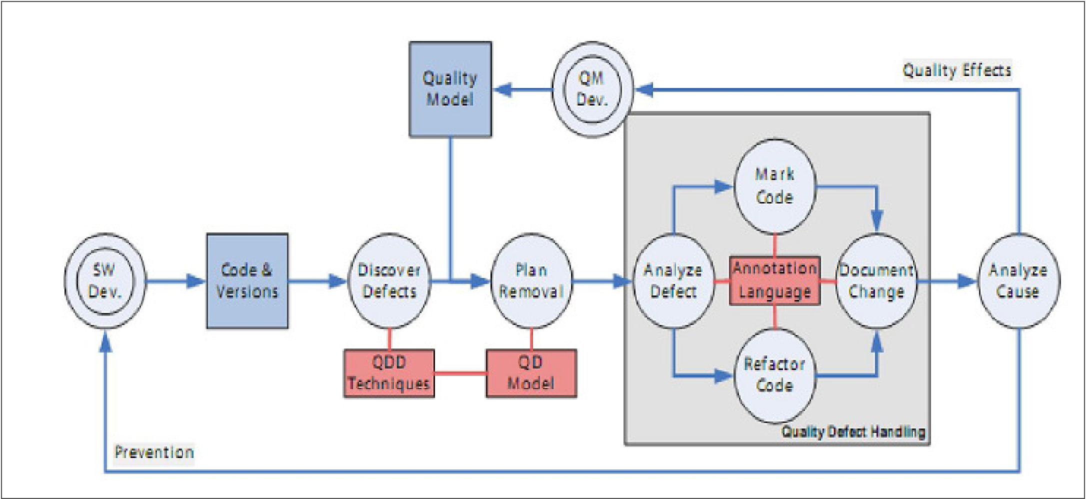


Figure 3. The quality defect discovery and handling process model

As the figure show, in the execution of the process, the following sub-process are performed :

Discover Defect : Manual or automatic quality defect discovery techniques are used to analyze the source code
Plan Removal : Based on the discovered, a sequential plan for the refactoring of the software system (or part) is constructed.
Analyze Defects : Process the list of potential quality defects, analyzes the affected software system (or part), and decides about the quality defect
Refactor Code : remove the quality defect from the software system.
Mark Code : Make where a potential quality defect is unavoidable or its removal would have a negative impact on an important quality.
Document Change : After the refactoring or marking, annotate with specific tags about the change or decision, and the experience about the activity
Analyze Cause : Statistics, information, and experiences about the existence of quality defects in the software systems

3.6. In Resume

To assure quality, agile software development organizations use activities such as refactoring between development iterations. Refactoring, or the restructuring of a software system without changing its behavior, is necessary to remove quality defects (i.e., bad smells in code, architecture smells, anti-patterns, design faws, software anomalies, etc.) that are introduced by quick and often unsystematic development. In this work, author described a process for the recurring and sustainable discovery, handling, and treatment of quality defects in software systems. The author come out with requirements for quality defect handling in agile software engineering such as : annotation language, tracking system or wiki, etc. The open door for this work is to built resarch ti increase automation of process in order to support team of developper with automated refactoring or defect discovery system.

4. QUALITY WITHIN AGILE MANAGEMENT

Agile methods is not only on the process model of software engineering, it is about management practicses also. Improving Quality by Exploiting Human Dynamics in Agile Methods[4] show how agile incoporate human factors, showed by theories and experiences to have critical impact on the succes rate of software production. Software engineering practices extensively involve humans under different roles (managers, analysts, designers, developers, testers, quality assurance experts, etc.) (Pfeeger, 2001; Sommerville, 2004). Authors in this paper deals with problem encourated at each level of management in an company applying eXtreme Programming, one of the most diffused agile method. The paper propose and discuss two models; the first model for personnel management based on the people CMM (The people capability maturity model [P-CMM] was developed by the software engineering institute [SEI] at Carnegie Mellon university (Curtis et al. 1995,2001)) and the next one proses a model that exploits developer personalities in pair programming.

4.1. Human Resource Management at the Corporate Level

Quality at the corporate level is to adress for example, workforce related problems such as bad staffing, inadequate training, bad competency, and performance management. People, people quality, and people management are essential for agile companies. As a consequence, Evaluation and Assessment people management models may help agile companies improve their people management processes and policies, assuring agile personnel quality.

People CMM, first published in 1995 and revised 2001 (version 2) (Curtis, Hefey, & Miller, 1995, 2001), is a five-level model that focuses on continuously improving the management and development of the human assets of a software systems organization.

The table below show the process areas of People CMM : Version 2.

MATURITY LEVEL	**FOCUS**	**KEY PROCESS AREAS**
5 optimizing	Continuously improve and align personal, workgroup, and organizational capabilty	- Continous workforce innovation - organizational performance alignement - continuous capability improvement
4 predictable	empower and integrate workforce competences and manage performance quantitatively	- mentoring - organizational capability management - quantitative performance management - competency-based assets - empowered workgroups - competency integration
3 defined	develop workforce competencies and workgroups, and align with business strategy and objectives	- patipatory culture - workgroup development - competency-based practices - career development - workforce planning - competency analysis
2 managed	managers take responsibility for managing and developing their people.	- compensation - training et development - performance management - work environment - communication and coordination staffing
1 initial	workforce practices applied inconsistently	(no kpa at this level)

Table 2. Process areas of the People CMM: Version 2

Maturity level 1 (Initial Level) : workforce practices are often ad hoc and inconsistent and frequently fail to achieve their purpose. Authors argue that XP organisation bypass the initial level cause "XP is a high disciplined methodology, thus organizations applying XP tend to retain skilled people, develop workforce practices, and train responsible individuals to perform highly co-operative best practices."

Maturity Level 2 (Key Process Areas at the Managed level) : At this level, P-CMM focus on establishing basic workforce practices and eliminating problems that hinder work performance. This capability is achieved by ensuring that people have the skills needed to perform their assigned work and by implementing the defned actions needed to improve performance (Training and development). XP addresses successfully training needs by rotating developers in pair programming and by involving them in significant practices such as planning game, testing, refactoring, and metaphor.

Maturity Level 3 (Key Process Areas at the Defined Level) : At this level, organization addresses organizational issues, developing a culture of professionalism based on well-understood workforce competencies. Competencis are designed to identify, develop, and use the knowledge, skills, and process abilities required by workforce to perform the organization's business activities, respectively. Authors agrue that XP organizations are well prepared to successfully address most of the P-CMM Level 3 cause " XP process establishes a high participatory culture (pair programming and other practices), spreading the fow of information within the organization, and incorporating the knowledge of developers into decision making activities, providing them with the opportunity to achieve career objectives."

Maturity Level 4 (Key Process Areas at the Predictable Level) : The key processes introduced in this level help organizations quantify the workforce capabilities and the competency-based processes it uses in performing its assignments. XP is a team-based process helping workgroups to develop more cohesion, capability, and responsibility. it requires that developers implement best practices in extreme levels using proven competency-based activities in their assignments. Managers trust the results that developers produce and the XP organization preserves successful results in its repository and exploits them as organizational assets.

Maturity Level 5 (Key Process Areas at the Optimizing level) : These practices cover issues that address continuous improvement of methods for developing competency at both the organizational and the individual level. Authors conclude by saying that "The results from measurements at level 4 and the culture of improvements established by the continuous implementation of the XP practices can help the XP organization to mature up to this level."

As a contribution to the P-CMM, authors proposed an adaptive P-CMM assessment process model for typcally XP-Organizations. The process model suggest is an adaptive people CMM assessment process model in the sense that the XP organization assesses itself against the process areas defned in each maturity level. The model is divided into three stages: Input, where the people process currently used by the XP organization and the adaptive people CMM framework are entered into the process. Operation, where the assessment process takes place. Output, where the results of the assessment process, in the form of a new improved process, are adopted by the people process management task and are communicated to the organization. The model is show below.

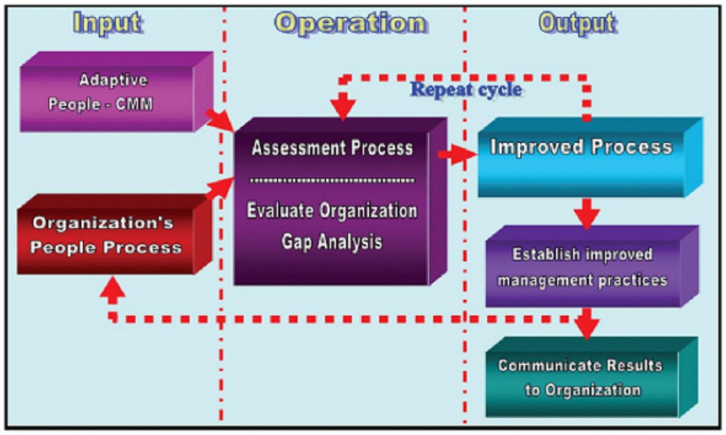


Figure 4. An Adaptive people CMM assessment process model for assessing XP- organizations

4.2. Quality at the Project/Team Level

Authors on this step of their work, went throw assessing and improving pair programming effectiveness based on developer personalities. The proof of human issues in pair programming was demonstrated when (beck,2000) said : "Extreme programming bases its software development process on a bunch of intensely social and collaborative activities and practices".
XP, is a disciplined practice in which the overall development activity is a joint effort, a function of how people communicate, interact, and collaborate to produce results.Two widely used tools to assist in the identifcation of personality and temperament types are the Myers-Briggs Type Indicator (MBTI) (Myers, 1975) and the Keirsey Temperament Sorter (KTS) (Keirsey et al., 1984). The result of their work first provide a table that summarise the salient characteristics of each personality type (list as : Extroverts, Introverts, Sensors, Intuitives, Thinkers, Feelers, Judges, Perceivers) and their suggestions for exploiting them in pair programming.

The table below give in short way summarize the temperaments salient characteristics and show authors suggestions for their use in pair programming.

Temperament Type	**Salient Characteristics**	**Suggested use in Pair Programming**
Artisans (SP) (Sensing-Perceiving)	Prefer concrete communications. Prefer a cooperative path to goal accomplishment. Possess a superior sense of timing. Prefer practical solutions and are lateral thinkers	Good as start-up persons and Effective brainstormers. May be good in decision making and May exhibit adaptability and be innovative
Guardians (SJ) (Sensing-Judging)	Prefer concrete communications. Prefer more a utilitarian approach. Are traditionalists and stabilizers Prefer rules, schedules, regulations, and hierarchy. Prefer that things remain as are	May be good in estimations (eg from the user stories). May be good in resource management. May be good in planning game, contracts. Are considered very responsible, succeed in assigned tasks
Idealists (NF) (Intuitive-Feeling)	Prefer more abstract communication. Prefer more a utilitarian approach. Prefer to guide others. Excellent communicators	Will contribute to pair spirit and morale. Are good in personal relationships. Are good in interaction with users and management. May be forward and global thinkers
Rationalists (NT) (Intuitive-Thinking)	Prefer more abstract communications. Prefer a cooperative path to goal accomplishment. Are natural-born scientists, theorists and innovators Posses highly valuing logic and reason. Prefer competence and excellence	Are good in subtask identification (eg in splitting user stories). Are good in long-range plan (ie planning game). Are good in analysis and design. Are considered good in inventing and configuring

Table 3. The salient characteristics of temperament types with respect to pair programming

Based on their findings, the results of their experiment, authors arrived with having the theory that considers pairs as adaptive ecosystems as framework. Their propose an adaptive pair formation/rotation process model as the figure below show (see Figure 5). The model can help organizations and managers build high-performance pairs out of talented developers. It describes three main phases: the setup phase, the assessment phase, and the improvement phase.

The setup phase includes the identification, understanding, and interpretation of the developer personalities—temperaments. The assessment phase includes a gap analysis and the construction or review of a set of guidelines and policies for pair formation/rotations. The improvement phase includes mini retrospectives (communication-collaboration reviews) for pair evaluation, and the establishment of the improved pair rotation process.

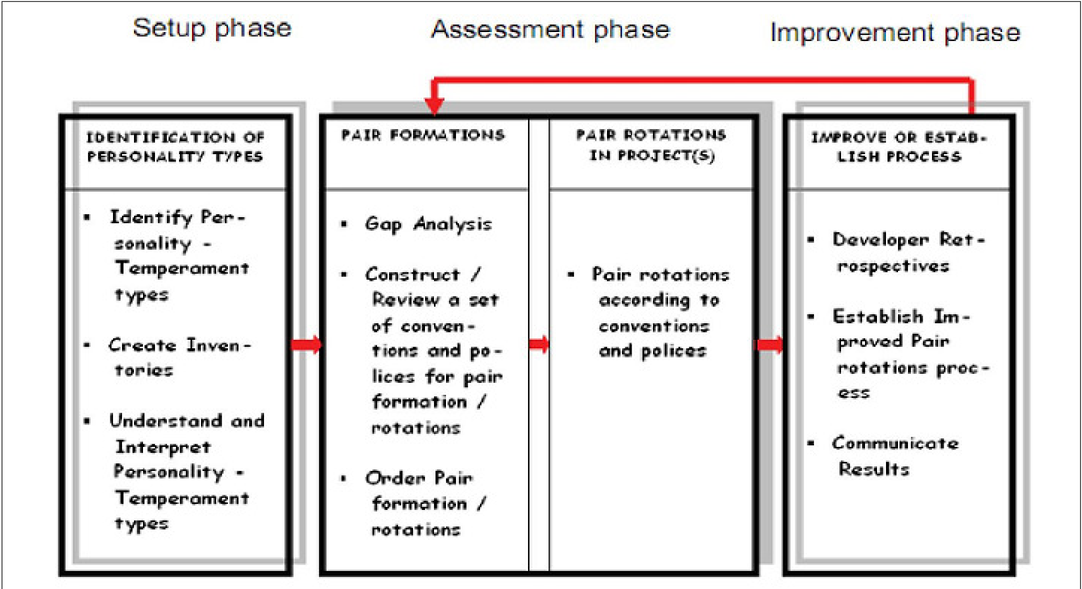

Setup phase
Assessment phase
Improvement phase
IDENTIFICATION OF PERSONALITY TYPES
Identify Personality - Temperament types
Create Inventories
Understand and Interpret Personality - Temperament types
PAIR FORMATIONS
Gap Analysis
Construct / Review a set of conventions and polices for pair formation / rotations
Order Pair formation / rotations
PAIR ROTATIONS IN PROJECT(S)
Pair rotations according to conventions and polices
IMPROVE OR ESTABLISH PROCESS
Developer Retrospectives
Establish Improved Pair rotations process
Communicate Results

Figure 5. AN adaptive pair formation/rotation process model

4.3. In Resume

Authors believed that organizations practicing XP should not have problems in addressing most of the PCMM level 2 and 3 KPAs (Key Process Areas). They described an adaptive people CMM assessment process model for assessing XP organizations and stepwise guidelines for its implementation. Also propose an adaptive pair formation/rotation process model, which identifies, interprets, and effectively combines developer variations. The proposed model can help organizations and managers improve pair effectiveness, by matching developers' personality and temperament types to their potential roles and tasks, effectively exploiting their differences in pair formations and rotations.

The more mature an organization, the greater its capability for attracting, developing, and retaining skilled and competent employees it needs to execute its business. Agile methods, in particular extreme programming, through their repeatable practices lead to an improved workforce environment with learning, training, and mentoring opportunities, improving workforce competencies.

5. AGILE QUALITY: FIELD EXPERIENCE

This chapter, is ending the survey paper with experiences from industry, comes from a large company, namely Siemens (USA). Siemens has gained as reported in the book of our survey, " in the past few years, considerable experience using agile process with several projects of varying size, duration, and complexity”. In the paper, quality improvement from using ADM_lessons learned [5]; authors report project in wich they have used agile process. The aim was to inform fellow agile developers and researchers about their methos for achieving quality goals, as well as providing an understanding of the current state of quality assurance in agile practices.

5.1. In-House Agile Processes

This part deals with the two agileprocess use in siemens.

The first process, named S-RaP[6] (an acronym for Siemens Rapid Prototyping), is a UI (user interface)-centered workfow-oriented approach that targets primarily the exploration of complex business requirements. The process evolved to provide rapid prototyping solutions for Siemens customers. The figure below shows this model.

S-RaP development is concentrated around two key artifacts:

- The Storyboard is the requirements and testing specifcation for the developers and a means to establish and communicate a common product vision among all stakeholders.

- Prototype provides the customer with a working representation of the fnal deliverable at an early stage. This gives the customer a hands-on testing experience with what has been developed, which helps to validate existing ideas and oftentimes generates new ones.

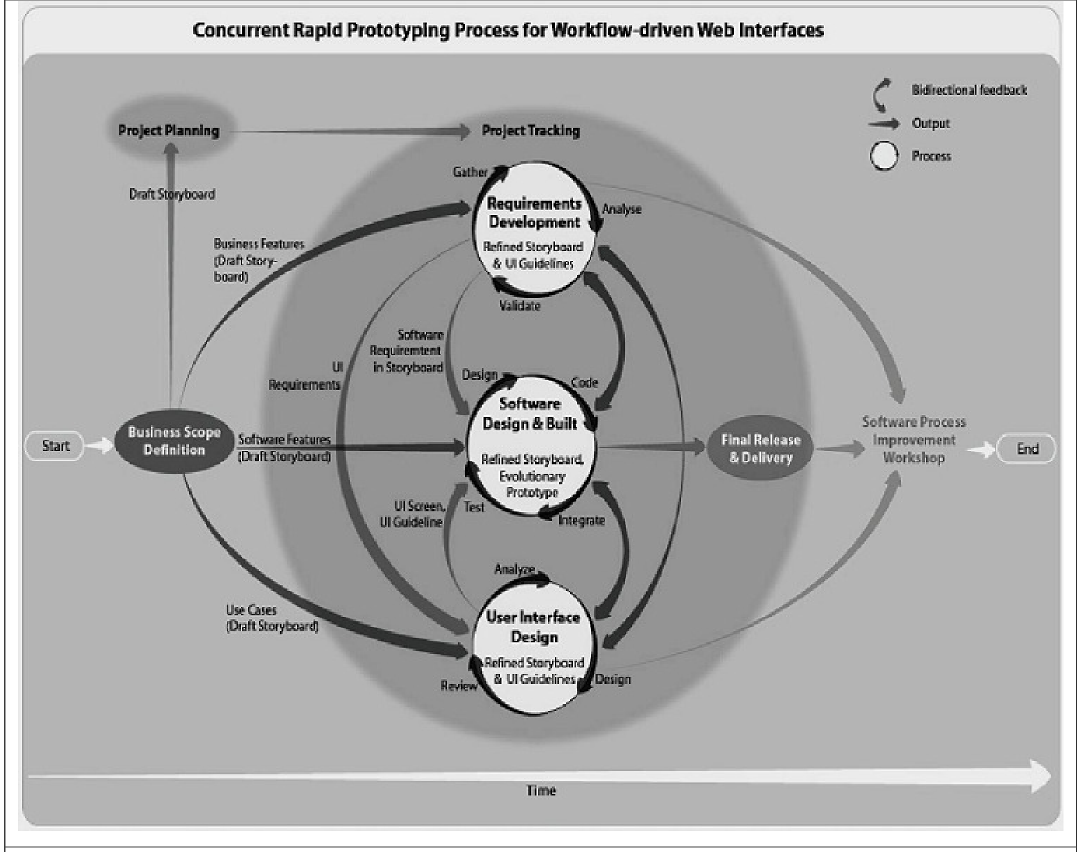
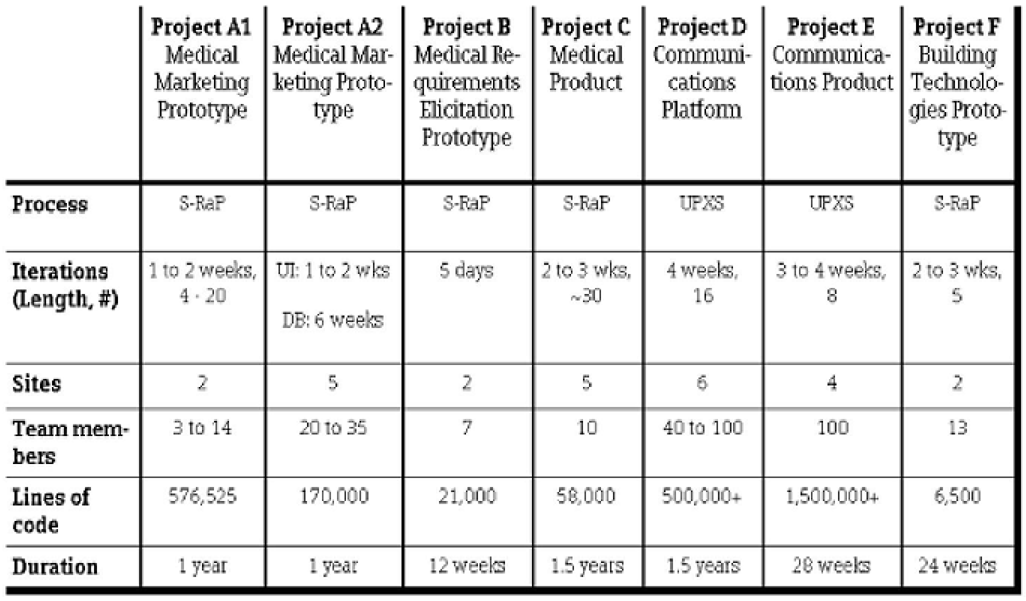

Concurrent Rapid Prototyping Process for Workflow-driven Web Interfaces
Bidirectional feedback
Output
Process
Project Planning
Project Tracking
Draft Storyboard
Business Features (Draft Story-board)
Gather
Requirements Development
Refined Storyboard & UI Guidelines
Analyse
Validate
Software Requirement in Storyboard
UI Requirements
Design
Code
Software Design & Built
Refined Storyboard, Evolutionary Prototype
Start
Business Scope Definition
Software Features (Draft Storyboard)
Final Release & Delivery
Software Process Improvement Workshop
End
UI Screen, UI Guideline
Test
Integrate
Analyze
Use Cases (Draft Storyboard)
User Interface Design
Refined Storyboard & UI Guidelines
Review
Design
Time

Figure 6. S-RaP process model (Nelson & Kim, 2004) (image by Kathleen Datta)

The second process, entitled UPXS[7], is a combination of traditional and agile practices (Unifed Process (Jacobson, Booch, & Rumbaugh, 1999), eXtreme Programming (Beck, 1999), and Scrum (Schwaber & Beedle, 2001)) that aims to support full-blown product development (even product-lines). Developed for a high-profle Siemens project, the process was designed to address the needs of a large distributed agile project. With a foundation of Scrum's team structure and activities, UPXS adds the project timeline model and phases of UP, along with iteration and task planning and development practices from XP. Similar to S-RaP, UPXS is executed in time-boxed iterations of 10 to 20 working days.

5.1.1. Project at Siemens

The processes previously described were employed in a number of different projects. The table below gives the characteristics of these projets taken at Siemens-USA.



	Project A1 Medical Marketing Prototype	**Project A2** Medical Marketing Prototype	**Project B** Medical Requirements Elicitation Prototype	**Project C** Medical Product	**Project D** Communications Platform	**Project E** Communications Product	**Project F** Building Technologies Prototype
Process	S-RaP	S-RaP	S-RaP	S-RaP	UPXS	UPXS	S-RaP
Iterations (Length, #)	1 to 2 weeks, 4 - 20	UI: 1 to 2 wks DB: 6 weeks	5 days	2 to 3 wks, ~30	4 weeks, 16	3 to 4 weeks, 8	2 to 3 wks, 5
Sites	2	5	2	5	6	4	2
Team members	3 to 14	20 to 35	7	10	40 to 100	100	13
Lines of code	576,525	170,000	21,000	58,000	500,000+	1,500,000+	6,500
Duration	1 year	1 year	12 weeks	1.5 years	1.5 years	28 weeks	24 weeks

Table 4. Siemens Project Characteristics

The numbers of team members and the duration for the project show that is was not a small project taken.

Project A1 and Project A2 : was an S-RaP project. The application ran in a Web browser and used simple HTML and JavaScript technologies. When the customer in Project A1 desired an advanced set of features that could not be easily done with its existing architecture, Project A2 was born.

Project B is a smaller S-RaP project that produced a prototype starting from a vague statement of customer needs.

Project C is another S-RaP project that produced a small 3-tier product. In terms of quality goals, the focus was initially on high security, so as not to compromise personal data, as well as a highly attractive and easy-to-use UI.

Project D is a UPXS project that began with a mostly centralized co-located team and has expanded into a worldwide-distributed project to develop a groundbreaking platform upon which future communications applications will run.

Project E is a UPXS project with a large number of distributed teams working on a product that will replace several legacy applications. The Web-based application interfaces with databases and communication hardware.

Project F is the smallest S-RaP project yet, which aimed to elicit, refne, and mature the requirements for a next generation product.

5.1.2 Quality Assurance : Common Goal

"QA (for agile methods) is looking at the same deliverables (as with plan-driven methods). But the process used to create the deliverables does affect how QA works" (McBreen, 2002). Siemens experiences have shown us that the cycle of customer involvement constant re-estimation, and constant reprioritization of scope and features is an inherent mechanism of agile methods that leads to high software quality.

Although each of our projects focused on their own set of quality goals, there were several common goals that were important to all of them. The 4 goals was applied to achieve one or more of siemens projects.

- Goal 1: The fnal deliverable should exhibit a high degree of correctness of implementation.
- Goal 2: The fnal deliverable is well suited to the expressed needs of the customer.
- Goal 3: The fnal deliverable is easy-to-understand, easy-to-learn, and easy-to-use.
- Goal 4: At any stage of development, code is easily analyzable, modifable, and extensible

5.1.3. Lessons learn.

The paper gives in the beautiful manner the lessons learn in applying agile methods to attempt the goal of quality at Siemens-USA. This is the list of the lessons.

Lesson 1: Use "living" documents whose lives are determined by the project's needs.
Lesson 2: Development needs to be proactive with the customer by providing solution alternates in a form easily grasped by the customer.
Lesson 3: Inexpert team members can be agile; however, the learning curve will be signifcant for those who lack technical expertise.
Lesson 4: Agile methodologies must be practiced with a culture of proactive communication to allow new members to acclimate quickly.
Lesson 5: Agile development needs agile project planning (Song et al., 2004).
Lesson 6: To achieve high customer satisfaction in agile development, collecting novice user feedback is just as important as regular customer feedback
Lesson 7: Collocation, when necessary, is best practiced within small teams (Song et al., 2004).
Lesson 8: Decomposing project tasks to assign different teams works best with vertical slices.
Lesson 9: Where practical, postpone refactoring until several instances of the part under consideration (component, behavior, etc.) are implemented.
Lesson 10: A high level of customer satisfaction can still be achieved, even if the resulting deliverables do not entirely meet expectations.

5.2. In Resume

The paper share an great experience in using customize agile project in Siemens. The implicit suggestions of the work was done in the section "the lessons learned" for improving QA in agile projects, authors feel that the most important is: " Actively attempt to capture and exploit informal communications". They suggested that it is important to identify quality goals early on in the project, even though they may change. The future work as an agile development group at Siemens Corporate Research is the interest in identifying metrics for measuring software quality in agile projects.

6. CONCLUSIONS

- Limitation of the work.

Software quality in agile development is not a straightforward topic. The survey study only the paper presented in Ioannis G et al. Book on Agile Software Development Quality Assurance. Even if this book is a reference in the domain, further paper must be study and an accent must be put on the most recent and pertinent work on the quality field.

- In resume

The first chapter on this work provides a review of the state-of-the-art of agile methodologies. However, it focuses primarily on the issue quality assurance. Then after the next chapter discusses refactoring, an agile procedure during which, among other activities, quality defect removal takes place. Because agile methods is not only on code writing but also on people interaction, the next chapter explores the management of the human resources that are involved in agile development; cause evidently human factors are critical for the success of agile methods. The last chapter resume the experiences of a large company, namely Siemens, with agile methodologies. This paper on study end with an summarize lessons learned from successes and failures while working for quality assurance in their projects..

ACKNOWLEDGEMENTS

The authors would like to thank everyone, just everyone!

REFERENCES

[1] Agile Software Development Quality Assurance. Ioannis G. Stamelos (Aristotle University of Thessaloniki, Greece) and Panagiotis Sfetsos (Alexander Technological Educational Institution of Thessaloniki, Greece). Release Date: February, 2007. Copyright © 2007.

[2] Agile Software Methods: State-of-the-Art; E. Mnkandla (Monash University, South Africa) and B. Dwolatzky (University of Witwatersrand, South Africa). p1-22.

[3] Handling of Software Quality Defects in Agile Software Development; Jörg Rech, Fraunhofer Institute for Experiemental Software Engineering (IESE), Germany.

[4] Improving Quality by Exploiting Human Dynamics in Agile Methods; Panagiotis Sfetsos, Alexander Technological Educational Institution of Thessaloniki, Greece; Ioannis Stamelos, Aristotle University, Greece.

[5] Quality Improvements from using Agile Development Methods: Lessons Learned. Beatrice Miao Hwong, Gilberto Matos, Monica McKenna, Christopher Nelson, Gergana Nikolova, Arnold Rudorfer, Xiping Song, Grace Yuan Tai,Rajanikanth Tanikella, Bradley Wehrwein.

[6] Gunaratne, Hwong, Nelson, & Rudorfer, 2004; Hwong, Laurance, Rudorfer, & Song, 2004; Nelson & Kim, 2004; Song, Matos, Hwong, Rudorfer, & Nelson, 2004; Song, Matos, Hwong, Rudorfer, & Nelson, 2005; Tai, 2005.

[7] Pichler, 2006; Smith & Pichler, 2005