ISSN 2047-3338

Agile Technology – A Novel Software Process Framework Analysis

V. Biksham[1], S. Rajeshwar[2], M. Naveen Kumar[3] and B. Nehru[4]
[1,2,3]Arjun College of Technology and Sciences, India
[4]Sri Venkateswara Engineering College, India

***Abstract*– Agile methodologies have become the mainstream of software development due to their enriched practices. Some commonly used practices include collaborative development, meeting evolving requirements with working software, simple design etc. These methods do not hold big upfront for early estimation of size, cost and duration due to uncertainty in requirements. It has been observed that Agile methodologies mostly rely on an expert opinion and historical data of project for estimation of cost, size and duration and also observed that these methods do not consider the vital factors affecting the same of project for estimation. The benefits of using Agile methodology are: i) in the absence of historical data and experts, existing agile estimation methods such as analogy, planning poker become unpredictable. We found that one-third of the study respondents use Agile methodologies to varying degrees, and most view it favorably due to improved communication between team members, quick releases and the increased flexibility of Agile designs. ii) It defines some alternative processes for projects in different situations. The main objective of this article is that it gives the novel process model of the agile Software deployment project. This paper helps to understand which steps are needed for initially deploying an Agile methodology and continuously improving the process.**

***Index Terms*– Process Models, Agile Process, XP, Scrum and Lean**

I. INTRODUCTION

AGILE Software Project Methodologies (ASP) [1] have been gaining acceptance among main stream software engineers since the late 1990s, Today agile projects are established to varying degrees in the educational, business, academic and software development professionals. We would like to understand how ASP methodologies are used, what kind of metrics and spread they have and what kind of success and demerits occur in each of these process. We believe strongly in describing empirical techniques to explore engendered by these research topics. While there is much to be learned from looking at the software measuring developer productivity and failure proneness, we can learn about their development artifacts, their practices and perceptions of development, and how the two are communicate.

Software development process is complex field with number of variables impacting the system. All software systems cannot be built with mathematical or physical certainty, so the systems are imperfect. Bridge building relies on physical and mathematical laws. Software Engineering, however, has no laws or clear certainties on which to build. As a result, software development is almost always flawed or sub-optimized. Also consider that the building blocks of software projects is programming languages, database platforms ,etc.), and the systems that act as building blocks contain bugs and cannot be relied on with certainty. Because the software development are inherently unstable and unreliable, organizations developing software must realize variables exist that are largely outside of management control. It is therefore fair to say that software development is more akin to new product research and development than it is to assembly-line style manufacturing. Software development is innovation, discovery, and artistry; each foray into a development project presents new and difficult challenges that cannot be overcome with one-size-fits-all, cookies cutter solutions.

Software Industry is the ability to accurate estimate its projects attributes, such as effort time cost is a priceless benefit in a highly-competitive global market. In order to attain useful estimate of these attributes, a set of software metrics must be implemented by the Industry.

On the other hand, over the last organizations of all types have replaced their traditional software development life cycles with a new set of process called agile methodologies. After almost a decade of the Agile Manifesto's publication [2] and much experimentation, agile methods have gained acceptances as effective software development methodologies and now widely used in many different types of industries. For however, the problem of successfully implementing software metrics is used in present.

If "working software is the primary measure of progress" as stated in the Agile Manifesto [6] then agile organizations need a method to measure completion of its products and determine report progress.

Measurement is a primary method for managing software life cycle works, which includes planning, controlling and monitoring of software projects. This is especially true in agile industries, where the activities are in "every day". However not all software metrics and standards developed over the years for traditional lifecycle models can be straight forward implemented by agile organizations without some adaptation. They need to follow different process in implementing a

successful metrics program, particularly if the industry is small organization.

II. SOFTWARE PROCESS MODELS

Software lifecycle has many process models. Here we discuss about the few process models:

A. The Waterfall Model

Waterfall model was proposed by Walker Royce in 1970's as an alternative to Build and Fix software development method in which code was written and debugged**.** System was not formally designed and there was no way to check the quality criteria.

Given below is a brief description of different phases of Waterfall model.

- *Feasibility Study:* It explores system requirements to determine project feasibility. Feasibility can be categorized into:

  i). Economic feasibility

  ii). Technical feasibility

  iii). Operational feasibility

  iv). Schedule feasibility

  v). Legal and contractual feasibility

  vi). Political feasibility

Economic feasibility is also called *cost-benefit analysis* and focuses on determining the project costs and benefits. Tangible costs and benefits are the ones which can be easily measured whereas intangible ones cannot be easily measured. Examples of tangible benefits are reduced errors, improved planning and control, reduced costs etc. Intangible benefits include timely information, better resource control, improved information processing, better assets utilization and many more. Economic feasibility uses the concept of time-value of money (TVM*)* which compares the present cash outlays to future expected returns.

Technical feasibility focuses on organization's ability to construct the proposed system in terms of hardware, software, operating environments, project size, complexity, experience of the organization in handling the similar types of projects and the risk analysis.

Operational feasibility deals with assessing the degree to which a proposed system solves business problems.

Schedule feasibility ensures that the project will be completed well in time.

Legal and contractual feasibility relates to issue like intellectual property rights, copyright laws, labor laws and different trade regulations.

Political feasibility finally evaluates how the key stakeholders within the organization view the proposed system (Jeffrey01).

B. Prototype

A prototype is a partially developed product. Robert T. Futrell and Shafer in their book Quality Software Project Management define prototyping as a process of developing working replica of a system (Robert02). Most of the users do not exactly know what they want until they actually see the product. Prototyping is used for developing a mock-up of product and is used for obtaining user feedback in order to refine it further as shown in Fig. 1.

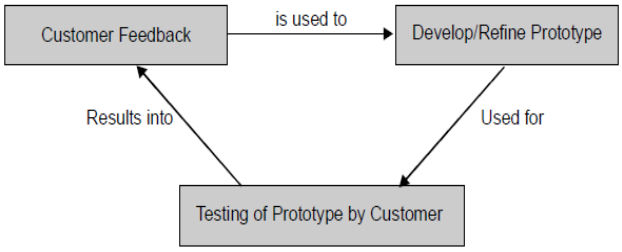

Fig. 1. Model of Prototype Process

In this process model, system is partially implemented before or during analysis phase thus giving the end users an opportunity to see the product early in the life cycle. The process starts by communicates the customers and developing the incomplete high level paper model. This document is used to build the initial prototype supporting only the basic functionality as desired by the customer. The prototype is then demonstrated to the customer for his/her feedback. After customer pinpoints the problems, prototype is further refined to eliminate the problems. This process continues till the user approves the rapid prototype and finds the working model to be satisfactory.

Prototype is classified into two approaches can be followed:

1). Rapid Throwaway Prototyping: This prototype is used for part of the systems where the development team does not have the understanding of the system. The quick and dirty prototypes are built, verified with the customers and thrown away. This process continues till a satisfactory prototype is built. At this stage now the full scale development of the product begins.

2). Evolutionary Prototyping: This approach is used when there is some understanding of the requirements. The prototypes thus built are not thrown away but evolved with time. Fig shows the concept of prototyping has led to the Rapid prototyping model and spiral model.

C. The Incremental Software Development Life Cycle Model

Software like all complex systems is bound to evolve due changing business requirements or new requirements coming up. The incremental software process development life cycle model is one of the popular evolutionary software process model used by industry. The model prioritizes the system requirements and implements them in groups. Each new release of the system enhances the functionality of the previously released system thereby reducing the cost of the project. Fig. 2 shows the working of the incremental model.

In the first release only the functionality A of the product is offered to the customer. Functionality A consists of core requirements which are critical to the success of the project. In the second release functionality A plus functionality B is offered and finally in release 3 functionality A, B as well as C is offered. Therefore, with each release in addition to incorporating new functionality in the system, functionality of earlier releases may also be enhanced. For example if a text

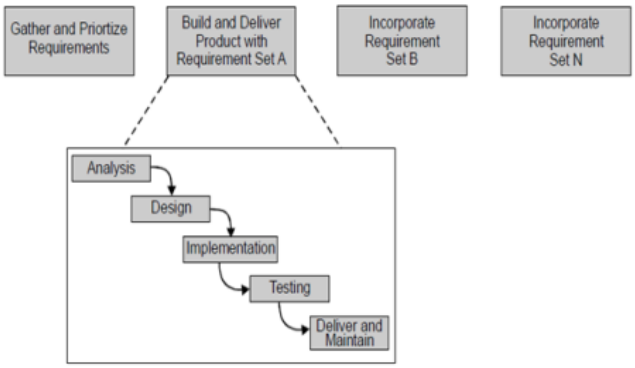

Fig. 2. Model of Incremental Process

editor is to be built, first version of the product will have basic editing facilities. The next version of the product can be released with enhanced editing facilities like formatting along with new features like spell checking. The incremental model is used when requirements are defined at the beginning of the project but at the same time they are expected to evolve over time. It can also be used for projects with development schedules more than one year or if deliveries are to be made at regular intervals.

D. The Spiral Model

Spiral model was proposed by Boehm in 1988 used for large size projects. The radial coordinate in the diagram represents the total costs incurred till date. Each loop of the spiral represents one phase of the development.

The model is divided into four quadrants. Each spiral represents the progress made in the project. In the first quadrant, objectives, alternative means to develop product and constraints imposed on the product are identified.

The next quadrant (right upper) deals with identification of risks and strategies to resolve the risks. The third bottom right quadrant represents the Waterfall model consisting of activities like design, detailed design, coding and test. With each phase after customer evaluates the product, requirements are further refined and so is the product. It is to be noted that number of loops through the quadrants are not fixed and vary from project to project.

The process starts with identification and prioritization of risks. A series of prototypes are developed for the risks identified (highest risk is considered first). For each

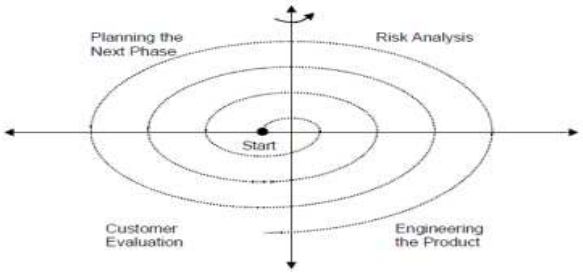

Fig. 3. Model of Spiral process

development cycle of the Spiral model shown in Fig. 3.

Spiral model is also termed as process model generator or meta model. For example if any project requirements are not clear models like Prototyping or Incremental can be derived from the spiral model.

E. The Rapid Application Development (RAD) Model

Rapid Application Development model was proposed by IBM in 1980's and software community by James Martin through his book Rapid Application Development. Feature of RAD model is increased involvement of the customer at all stages of life cycle through development tools.

If the requirements of the software can be modularized in such a way that each of them can be completed by different teams in a fixed time. RAD model is quick turnaround time from requirements analysis to the final delivered system. The time frame for each delivery is normally 60 to 90 days called time box which is achieved by using tools like Visual C++, JAVA, XML, .NET etc.

The RAD model consists of following four phases:

Requirements Planning– focuses on collecting requirements using elicitation techniques like brainstorming,

User Description– Requirements are detailed by taking users feedback by building prototype using development tools.

Construction– The prototype is refined to build the product and released to the customer.

Cutover– involves acceptance testing by the user and their training.

The process therefore starts with building a rapid prototype (a working model which is functionally equivalent to a subset of final product) and is delivered to customer for use and his feedback. Once the user/customer validates the rapid prototype after using it, Requirement Specification Document is derived and design is done to give final shape to the product. After the product is installed, maintenance of the product is continued by refining the requirements, specification, design or coding phase.

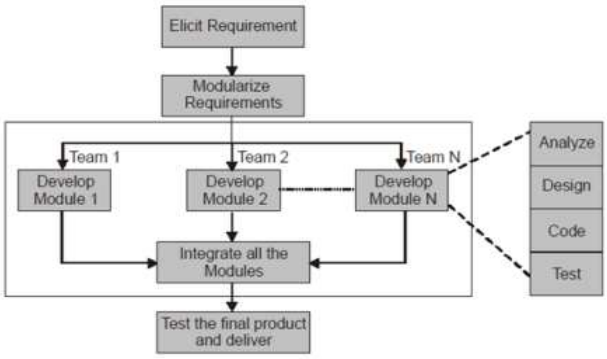

Fig. 4. Model of Rapid Application Development Process

III. Agile Process and Project Management

XP and Scrum iterative approach process models are widely used in the Agile Philosophy.

A. XP

Extreme programming(XP), concentrates on the development rather than managerial aspects of software projects. XP was designed so that organizations would be free to adopt all or part of the methodology.

XP projects start with a release planning, followed by several iterations, each of which concludes with user acceptance testing. When the products has enough features to satisfy users, the team terminates iteration and releases the software. "User stories" to describe need of the software should fulfill. User stories help the team to estimate time and resources necessary to build the release and to define the user accepteance tests. User representative part of the XP team can add detail requirements as the software is being built. This allows requirements to evolve as both users and developers define what the product will look like.

In release plan, team breaks up the development tasks into iterations. Release plan defines each iteration plan, which drives the development for that iteration. At the end of iteration, conduct the acceptance tests against the user stories. If they find bugs, fixing the bugs becomes a step in the iteration. If users decide that enough user stories have been delivered, the team can choose to terminate the project before all of the originally planned user stories have been implemented.

Fig. 5 shows a simplified version of XP. Full XP includes many steps in release planning, iteration, and acceptance.

1). Integrate Often: Development teams must integrate changes into the development baseline at least once a day.

2). Project Velocity: Velocity is a measure of how much work is getting done on the project. This important metric drives release planning and schedule updates.

3). Pair Programming: All code for a production release is created by two people working together at a single computer. XP proposes that two coders working together will satisfy user stories at the same rate as two coders working alone, but with much higher quality.

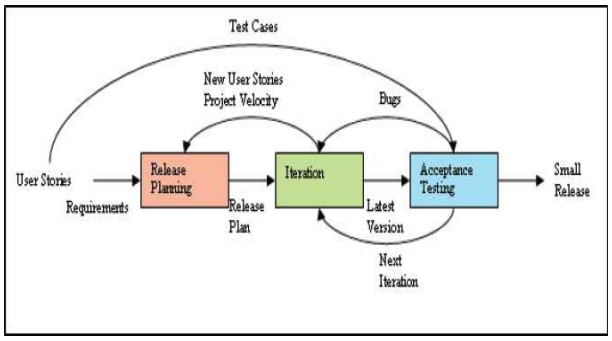

Fig. 5. XP Process Model

4). User story: A user story describes problems to be solved by the system being built. These stories must be written by the user and should be about three sentences long. User stories do not describe a solution, use technical language, or contain traditional requirements-speak, such as "shall" statements. Instead, a sample user story might go like this: Search for customers. The user tells the application to search for customers. The application asks the user to specify which customers. After the user specifies the search criteria, the application returns a list of customers meeting those criteria. Because user stories are short and somewhat vague, XP will only work if the customer representative is on hand to review and approve user story implementations. This is one of the main objections to the XP methodology, but also one of its greatest strengths.

B. Scrum

Scrum is the term for a huddled mass of players engaged with each other to get a job done. In software development, the job is to put out a release. Scrum for software development came out of the rapid prototyping community because prototypes wanted a methodology that would support an environment in which the requirements were not only incomplete at the start, but also could change rapidly during development.

Scrum project is a backlog of work to be done. This backlog is populated during the planning phase of a release and defines the scope of the release. After the team completes the project scope and high-level designs, it divides the development process into a series of short iterations called sprints. Each sprint aims to implement a fixed number of backlog items. Before each sprint, the team members identify the backlog items for the sprint. At the end of a sprint, the team reviews the sprint to articulate lessons learned and check progress. During a sprint, the team has a daily meeting called a scrum. Each team member describes the work to be done that day, progress from the day before, and any blocks that must be cleared. To keep the meetings short, the scrum is supposed to be conducted with everyone in the same room—standing up for the whole meeting. When enough of the backlog has been implemented so that the end users believe the release is worth putting into production, management closes development. The team then performs integration testing, training, and documentation as necessary for product release.

Scrum development process concentrates on managing sprints. Before each sprint begins, the team plans the sprint, identifying the backlog items and assigning teams to these items. Teams develop, wrap, review, and adjust each of the backlog items. During development, the team determines the changes necessary to implement a backlog item. The team then writes the code, tests it, and documents the changes. During wrap, the team creates the executable necessary to demonstrate the changes. In review, the team demonstrates the new features, adds new backlog items, and assesses risk. Finally, the team consolidates data from the review to update the changes as necessary.

The entire team, including management, users and interested parties demonstrates progress from the sprint and

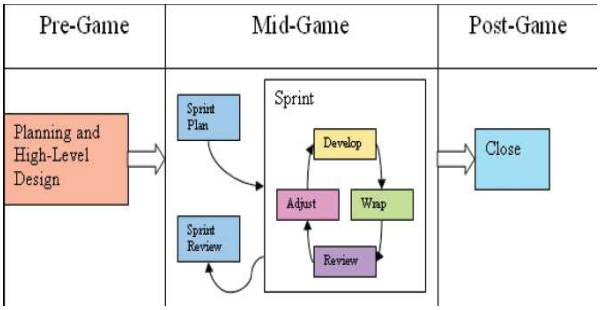

Fig. 6. SCRUM Process Model

reviews the backlog progress. The team reviews the remaining backlog and adds, removes or reprioritizes items as necessary to account for new information gathered during the sprint.
The concepts of the Scrum are the following:

1). Burn down char: This chart, updated every day, shows the work remaining within the sprint. The burn down chart is used both to track sprint progress and to decide when items must be removed from the sprint backlog and deferred to the next sprint.

2). Product Backlog: Product backlog is the complete list of requirements including bugs, enhancement requests, and usability and performance improvements that are not currently in the product release.

3). Scrum Master: The Scrum Master is the person responsible for managing the Scrum project. Sometimes it refers to a person who has become certified as a Scrum Master by taking Scrum Master training.

4). Sprint Backlog: Sprint backlog is the list of backlog items assigned to a sprint, but not yet completed. In common practice, no sprint backlog item should take more than two days to complete. The sprint backlog helps the team predict the level of effort required to complete a sprint.

C. Agile Lean Software Process Model

Basically the lean development (manufacturing) principles can also be applied to the software development process to resolve the issues and to improve the process and obtain good results.

1). Eliminate Waste: lean first step is to understand what "value" is and what activities and resources are absolutely necessary to create the value. Since no one wants to consider what they do as waste, the job of determining what value is and what adds value is something that needs to be done at a fairly high level. The seven types of developing wastes are illustrated in Table 1.

The seven wastes of software development, based on the above framework, are illustrated below in Table 2.

The project teams should try to avoid these wastes, which are very commonly produced during the development process.

2). Incorporate Feedback

Incorporate feedback does not mean to "Freeze the Specs." On the contrary, product (and software project) specifications change constantly. Lean discipline demands instantaneous adaptation to changing market conditions, which

Table 1: Shows the seven wastes of manufacturing are also applicable to software development

The Seven Wastes of Manufacturing
Overproduction
Inventory
Extra processing steps
Motion
Defects
Waiting
Transportation

Table 2: Shows the seven wastes of software development

The Seven Wastes of Software Development
Overproduction (Extra features)
Inventory (Requirements)
Extra processing steps (Extra Steps)
Motion (Finding Information)
Defects (Bugs not caught by tests)
Waiting (Waiting for decisions, including customers)
Transportation (Handoffs)

is best affected with a flexible architecture that readily accommodates changes, monitoring techniques that detect errors before they occur, and tests that are designed before development begins.

The Incorporate feedback rule has been widely used to justify the decision to develop a detailed system design before code is written. The problem with this approach lies in the assumption that customer requirements are static and can be defined by a predetermined system. Because requirements do change, and frequently throughout the life cycle of most systems, they cannot be adequately fulfilled by a rigid design. If we acknowledge the axiom that customers may not know what they want at the beginning of development and that their needs might change midstream, we must incorporate a method of obtaining customer feedback during development. Instead, most software development practices include a complex "change control process" that discourages developers from responding to user feedback. Far from ensuring a quality result, these change-resistant processes actually get in the way of "Doing It Right". Lean development employs two key techniques that make the change easy.

Just as Lean Production builds tests into the manufacturing process to detect when the process is broken, similarly Lean development must "build tests" at various stages of the development process. As development proceeds and changes are made, the unit and regression tests are run. If the tests don't pass, programming may be stopped until the problem is found and corrected. A comprehensive testing capability is the best way to accommodate change throughout the development process.

The second technique that facilitates change is “refactoring” (Improving the design without changing functionality), or improving the design of existing software in a controlled and rapid manner. With refactoring, initial designs can focus on the basic issue at hand rather than speculate about other features that may be needed in the future. Later in the process, refactoring techniques can incorporate these additional features, as they are required, making it easy to accommodate the future if and when it becomes the present.

3). Empower those who add value: A basic principle of Lean Production is to drive decisions down to the lowest possible level, delegating decision-making tools and authority to the people "on the floor." Often when software development environments under-perform, the instinctive reaction is to impose more rigid processes, specifying in greater detail how people should do their jobs. Lean Production principles suggest exactly the opposite approach. When there is problem in manufacturing, a team of outside experts is not sent in to document in more detail how the process should be run. Instead, people on the manufacturing floor are given tools to evaluate and improve their own areas. They work in collaborative teams with the charter to improve their own processes and the links to nearby processes for which they are suppliers or customers. Their supervisors are trained in methods of forming and encouraging work teams to solve their own problems.

Lean software development similarly gives priority to people and collaborating teams over paper work and processes. It focuses on methods of forming and encouraging teams to address and resolve their own problems, recognizing that the people doing the work must determine the details. Software development involves the handoff of information at least once (from user to programmer) and often more than once (from user to designer to programmer). One school of thought holds that it's best to transfer all such information in writing, but in fact, a great amount of tacit knowledge is lost by handing off information on paper. A second school of thought believes that it’s far more effective to have small collaborating teams work across the boundaries of an information handoff, minimizing paperwork and maximizing communication.

4). Continuous Culture improvement: In many software development projects today, excellence means the ability to adapt to fast-moving, rapidly changing environments. Process-intensive approaches such as the higher levels of Software Engineering Institute's (SEI) Capability Maturity Model (CMM) may lack the flexibility to respond rapidly to changes and more over these process-documentation programs indicate excellence only when the documented process excels in the context of its use. So these accreditations have their own advantages and disadvantages. Iterative development can effectively employ the Plan-Do-Check-Act method. During the first iteration, the handoff from design to programming or programming to testing may be a bit rough. It's ok if the first iteration provides a learning experience for the project team, because the subsequent iterations will allow the team to improve its process.

In an iterative project environment becomes an operational environment, because processes are repeated and Deming's techniques of process improvement can be applied from one iteration to the next. A simple Plan-do-check-act principle can be followed:

- *Plan:* Choose a problem. Analyze it to find a probable cause.
- *Do:* Run an experiment to investigate the probable cause.
- Check: Analyze the data from the experiment to validate the cause.
- *Act:* Refine and standardize based on the results.

Product/Solution improvement is also enhanced in the iterative process, particularly if refactoring is used. In fact, refactoring provides a tremendous vehicle to apply the principle of continuous improvement to software development. However, need an improvement model that can span more than a single project. Must improve future project performance by learning from existing ones. Here again, Lean production can point the way.

5). Customer Requirements: Philip Crosby defines quality as "conformance & Performance to requirements." The 1994 Standish Group study "Charting the Seas of Information Technology—Chaos" stated that that the most common cause of failed projects is missing, incomplete or incorrect requirements. The software development world has responded to this risk by amplifying the practice of gathering detailed user requirements and getting user signoff prior to proceeding with system design. However, this approach to defining user requirements is deeply flawed. As discussed above in the incorporate feedback principle the process should have the provision for the customer to make changes. The Software compliance and User (Customer) acceptance testing must be done with reference to the customer requirements.

6). Demand: Lean Software development means rapid, Just-in-Time delivery of value. In manufacturing, the key to achieving rapid delivery is to manufacture in small batches pulled by a customer order. Similarly in software development, the key to rapid delivery is to divide the problem into small batches (increments) pulled by a customer test. The single most effective mechanism for implementing lean production is adopting Just-in-Time, Pull from demand flow. Similarly, the single most effective mechanism for implementing lean development is delivering increments of real business value in short time-boxes.

7). Flow: In lean software development, the idea is to maximize the flow of information and delivered value. In lean production, Maximizing flow does not mean automation. Instead it means limiting what has to be transferred, and transferring that as few times as possible over the shortest distance with the widest communication bandwidth. Similarly in software development the idea is to eliminate as many documents and handoffs as possible. In the lean software development emphasis is to pair a skilled development team with a skilled customer team and to give them the responsibility and authority to develop the system in small, rapid increments, driven by the customer priority and feedback.

8). Project Scope: Project managers have been trained to focus on managing scope, just as in manufacturing production managers concentrated on maximizing machine productivity.

However, Lean software development is fundamentally driven by time and feedback. In the same way that localized productivity optimization weakens the overall manufacturing process, so focusing on managing scope impairs project development.

Holding the scope to exactly what was envisioned at the beginning of a project offers little value to the user whose world is changing. In fact, it imparts anxiety and paralyzes decision-making, ensuring only that the final system will be outdated by the time it's delivered. Managing to a scope that's no longer valid wastes time and space, necessitating inefficient issue lists, extensive trade-off negotiations and multiple system fixes. However, as long as limiting a project to its original scope is a key project management goal; local optimization will flourish at the expense of the project's overall quality.

9). Quality Supply: High quality and creativity of the supply chain partnerships far outweighed the putative benefits of competitive bidding and rapid supplier turnover. Partner companies helped each other improve product designs and product flows, linking systems to allow just-in-time movement of goods across several suppliers with little or no paperwork.

IV. COMPARATIVE STUDY

Agile methods are on the adaptive side of this continuum. Adaptive methods focus on adapting quickly to changing realities. When the needs of a project change, an adaptive team changes as well. An adaptive team will have difficulty describing exactly what will happen in the future. The further away a date is, the vaguer an adaptive method will be about what will happen on that date. An adaptive team cannot report exactly what tasks are being done in the next week, but only which features are planned for next month. When asked about a release six months from now, an adaptive team may only be able to report the mission statement for the release, or a statement of expected value and cost.

Predictive methods, in contrast, focus on planning the future in detail. A predictive team can report exactly what features and tasks are planned for the entire length of the development process. Predictive teams have some problems in changing direction. The plan is typically optimized for the original destination and changing direction can require completed work to be started over. Predictive teams will often institute a change of control board to ensure that only the most valuable changes are considered.

Agile project management has many process models compare to other process model. Agile scrum, Xp, are the software development lifecycle but novel lean development process is the concept of manufacturing process which is efficient in products, the same process for software development is most effective in projects.

V. CONCLUSIONS

Agile software development stresses rapid iterations, small and frequent releases, and evolving requirements facilitated by direct user involvement in the development process. Agile application lifecycle management tools provide a framework to visualize scope, orchestrate mundane and repetitive development tasks, and enforce process. Unlike agile-specific products offered by agile-only vendors, and can be applied equally well to agile as well as more traditional serial development processes, so they can support all the development activities within an enterprise. In this article proposes the development of novel lean development process, lean steps taken from the manufacturing products and comparing the agile methods, scrum xp with lean. These vital factors include mainly; project domain, configuration, performance, complex processing, data transaction, operation ease, multiple sites and security. Future work of this article is extended with the requirements and metrics of projects, which we need to implement in the lean software development process.

REFERNCES

[1] Mufazzal Badani, “Mapping Agile development practices to Traditional PMBOK”, June, 4th 2001, www.pmiissig.org/pds/DOCS/BadaniBioandAbstractMapping AgileDevelopmentPractices.doc, accessed 29 Feb 2007.

[2] David F. Rico, “Agile Methods and the Links to Customer Satisfaction and Firm Performance”, September, 2006. P:11.

[3] Orlicky, “The Successful Computer System: Its Planning, Development, and Management in a Business Enterprise”, McGraw-Hill, New York, 1969.

[4] Basili and Turner, “Iterative enhancement: A practical technique for software development. IEEE Transactions on Software Engineering “,1975, Pages 390-396.

[5] Zarraga and Bonache, “The impact of team atmosphere on knowledge outcomes in self managed teams “. Organization Studies, 26(5), Pages 661-681, 2005.

[6] Babchuk, N., & Goode, W. J. (1951). “Work incentives in a self determined group“. American sociological Review, 16(5), Pages 679-687, 1951.

[7] Ahituv, Hadass, & Neumann, “A flexible approach to information system development “, MIS Quarterly, 8(2), Pages 69-78 ,1984.

[8] Sanjiv Augustine and Susan Woodcock, “Agile Project Management Pace Systems”, CC Pace, www.ccpace.com, par. 2, page 5- 2003 , Accessed 12 Dec 2006. Published in PM World Today - May 2007 (Vol. IX, Issue V) PM.

[9] Ken Schwaber, Mike Beedle, “Agile Software Development with Scrum”, Prentice Hall, 2001, pp. 100-101.

[10] In fact, it is debatable whether Quality is really an adjustable factor. As professionals, software developers find it very objectionable when asked to skimp on quality. Surgeons or lawyers would be sued for malpractice, and for the same ethical implications software developers resent this charge. Software developers should always aim for high quality software, period.

[11] Kent Beck, “Extreme Programming Explained: Embrace Change”, Addison-Wesley, 2000, pp. 15-19.

[12] Black. S. E. Boca., P. P.; Bowen. J.P.; Gorman, J.; Hinchey. M.G. (September 2009). "Formal versus agile: Survival of the fittest". IEEE Computer 49 (9): 39–45.

[13] Boehm. B.; R. Turner (2004 Balancing Agility and Discipline: A Guide for the Perplexed. Boston, MA: Addison-Wesley. ISBN 0-321-18612-5. Appendix A, pages 165-194.

[14] Ken Schwaber, Mike Beedle, “Agile Software Development with Scrum”, Prentice Hall, 2001.

V. Biksham, Asst. Prof. at Arjun College of Tech & Sciences, M.Tech (SE) from Aurora Technological & Research Institute (JNTUH), B.Tech (CSE) from SRTIST (JNTUH). His areas of interest are Computer Networks, Software Engineering and Information Security.

S. Rajeshwar, Asst. Prof. at Arjun College of Tech & Sciences, M.Tech (CSE) from Nagarjuna University, Guntur B.Tech (CSE) from SRTIST (JNTUH). His areas of interest are Compiler Design, Database Management Systems, Automata Theory, and Software Engineering.

M. Naveen Kumar, Asst. Prof. at Arjun College of Tech & Sciences, M.Tech from HITS (JNTUH), B.Tech from Aditya Engineering college (JNTU). His areas of interest are Computer Networks, Operating System, Information Security and Software Engineering.

B. Nehru, Asst. Prof. at Sri Venkateswara Engineering College, M.Tech from Aurora Technological & Research Institute (JNTUH), B.E from Vasavi Engg College (OU). His areas of interest are Advanced Data Structures, Object Oriented Programming and Software Engineering.