Volume 4, Issue 6, June 2014 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Impact of Utility trees and Quality attributes on evaluating Software Architecture Evaluation Methodologies in Agile Adaptation

Jyoti Budakoti
B.Tech in Computer Science & Engineering,
G.B. Pant Engineering College, Pauri Garhwal,
Uttarakhand, India

Amit Singh Gaur
B.Tech in Electronics & Communication Engineering,
G.B. Pant Engineering College, Pauri Garhwal,
Uttarakhand, India

Abstract: ***Software architecture methodologies are the key business asset for an organization because architectures are complex and involve many design tradeoffs analysis of the underlying methodologies. Several studies have shown that 50% to 70% of the total lifecycle cost for a software system is spent on evolving the system. Incorporating anticipated changes generally requires considerably less effort than unanticipated modifications, thus it is significantly important to prepare a software system for likely adaptable to changes during system development. The role of software architecture plays a considerable significance in achieving this, but without undertaking a formal analysis procedure, the organization cannot ensure about the architectural decisions —particularly those which affect the achievement of quality attributes such as performance, modifiability ,availability and security—are advisable ones that appropriately mitigate risks and development costs. In this paper, we propose an analysis method for software architecture methodologies in agile in which we will discuss some of the technical and organizational foundations for performing architectural analysis in agile adaptation, and reflect iterative development approaches that in turn leverage agile methods.***

Keywords: Software Architecture Methodologies, Traditional Methodologies, Agile Adoption, Agile Architecture, Agile Techniques, Quality Attributes, Utility Trees

I. INTRODUCTION

The world around most software systems is constantly changing. This requires software systems to be modified several times after their initial development. A number of studies [1, 6] have shown that 50%to 70% of the total lifecycle cost of a software system is spent after initial development. One of the reasons for the high cost of software maintenance is that productivity of changing existing code is at least an order of magnitude lower than developing new software or relatively independent extensions to an existing system. Due to this, incorporating anticipated changes, changes for which the software system has been prepared, generally requires considerably less effort and cost than including unanticipated changes. Preparing a software system for likely changes is an activity that takes place during the development of a software system design and specially, the software architecture of the system plays an important role in achieving this. Architecture is the key ingredient in a business or an organization's technological success. A system is motivated by a set of functional and quality goals. For example, if a car manufacturer is creating a new car, you will only know the car's performance when you actually drive it and feel it, and not just by reading specifications or by asking questions. Similarly, to evaluate the architecture, apply it to the specific functional scenarios and measure the quality attributes. But if it is to be successful, it must do so within strict performance, modifiability, availability, and cost parameters of architecture. The architecture is the key to achieving—or failing to achieve—these goals. Having a structured method helps ensure that the right questions regarding architecture will be asked *early*, during the requirements and design stages when discovered problems can be solved relatively cheaply. It guides users of the method—the stakeholder—to look for conflicts and for resolutions to these conflicts in the software architecture designing. This method has also been used to analyze legacy systems. This frequently occurs when the legacy system needs to support major changes, integration with other systems or other significant upgrades. The ATAM draws its inspiration and techniques from three areas: the notion of architectural styles; the quality attribute analysis communities; and the Software Architecture Analysis Method (SAAM) [Kazman 94] .Efforts on cataloguing the implications of using design patterns and architectural styles contribute, frequently in an informal way, to ensuring the quality of a design [Buschmann 96]. More formal efforts also exist to ensure that quality attributes are addressed. These consist of formal analyses in areas such as performance evaluation [Klein 93], Markov modeling for availability [Iannino 94], and inspection and review methods for modifiability [Kazman 94].But these techniques, if they are applied at all, are basically typically applied in case of isolation and their implications are considered in isolation. It is risky because all design involves tradeoffs and if we simply optimize for a single quality attribute, we stand the chance of ignoring other attributes of importance and even

more significantly, if we do not analyze architecture for multiple attributes, we have no use of understanding the tradeoffs made in the architecture—places where improving one attribute causes another one to be compromised.

Based on an understanding of the expected future evolution of the software system, the software architect can employ various design solutions to prepare for the future incorporation of new and changed requirements. One problem, however, is that the software architect has few means or techniques available for determining whether the goal of high modifiability has been achieved by the set of employed design solutions, either in terms of predicted maintenance cost or with respect to avoiding inflexibility in the current design of architecture. Similarly, fewer methods and techniques are available for selecting among a number of alternative architectures to be used for a system. One area of research addressing the above is software architecture analysis. In software architecture analysis, the software architecture of a software system is analyzed to predict one or more quality attributes. At present, a number of methods for software architecture analysis exist which includes SAAM [7], architecture level prediction of maintenance [8] and inflexibility assessment [9]. Although these methods do share a number of similarities, however, there are basic differences as well.

I discussed some of the architecture evaluation methods in above paragraph. In short, the idea is - Rather than just using a checklist for technical evaluation, we should test drive each functional scenario against architectural decisions and judge the impact on quality attributes (utility tree) and evaluate the architecture. Such a review will be more specific to the project and hence, more beneficial. Many accept the advantages of these methods, however, often criticize these methods for their overhead. I think, these methodologies can certainly be adapted for agile use. We do not have to stick to the elaborate process that software architecture has recommended, but instead, we should adapt it for our use by sticking only to its principles. In fact, in my experience, these methodologies are more beneficial if we use them in conjunction with agile development methodologies.

In Agile world, the application functionality is to be built in agile way, with constant visibility to business, and frequent changes to the features. However, the same is not the case for its architecture. If the architecture is allowed to evolve with changing requirements, it causes frequent rework, constant re-factoring and in fact, it counters the 'Agile' approach. It makes sense to spend upfront time on analyzing architecture evaluation and ensure that it is adaptable to accommodate the future enhancements and modifications.

In this paper, we present a generalized, adaptable agile approach for software architecture analysis. The agile approach depends on the target requirement; the method is adapted by using different techniques in some of the main steps. The method will be useful to a number of industrial cases, including software architectures at medicals, marketing, telecommunications, medicals, logistics, embedded systems and administrative information systems and many more.

II. EXISTING SOFTWARE ARCHITECTURES METHODOLOGIES

A. ATAM (Architecture Trade-off Analysis Method)

This methodology analyzes how well the software architecture satisfies the quality attributes (e.g. scalability, modifiability, performance etc), by applying the architecture to short-listed functional scenarios. It prescribes developing a quality attribute utility tree, and analyzing it for each scenario and alternative architecture approaches. For each scenario, the method prescribes identifying the following:

1) Sensitivity points – a collection of components that are critical for achieving a quality attribute,
2) Trade-off points – a sensitivity point that affects multiple quality attributes, typically trades one off for the other,
3) Risks – something that inhibits the system from achieving its quality goal (this also includes the decisions that are not taken),
4) Non-risks – something that is done right (and should not be changed)

B. CBAM (Cost Benefit Analysis Method)

CBAM begins where ATAM leaves off. It prescribes analyzing the cost, benefits and schedule implications as well for architectural approaches before making the final decisions.

C. ARID (Active Reviews for Intermediate Design)

This is more of a design review than architecture review, but uses the same principle of applying design to scenarios, and even writing a pseudo code to evaluate different parts of design.

The SEI has documented a very formal step by step process for all these methods. One way that may be counterproductive as people tend to focus on activities, rather than the principles behind these methods. There is a great scope to tailor these methods to suit your organization, and conduct such evaluation in agile way.

III. ROLE OF ADAPTIVE AGILE METHODOLOGY

For over a decade now, there has been an ever increasing variety of agile methods available includes a number of specific techniques and practices of software development. Agile methods are a subset of ―iterative and evolutionary methods‖ [10, 11] and are ―based on iterative enhancement‖ [12] and ―opportunistic development processes‖ [13]. Majority of agile development methods promote development, collaboration, teamwork and process adaptability throughout the life-cycle of the project [16]. The major methods include eXtreme Programming (Beck, 1999), [14], Scrum (K. Schwaber &

Beedle, 2002), [15], Dynamic Systems Development Method (Stapleton, 1997), Adaptive Software Development (Highsmith, 2000), Crystal (Cockburn, 2002), and Feature-Driven Development (Palmer & Felsing, 2002). [17], [18], [19], [20]. Agile software development methodology process flow (Scrum) has been shown in figure 1.

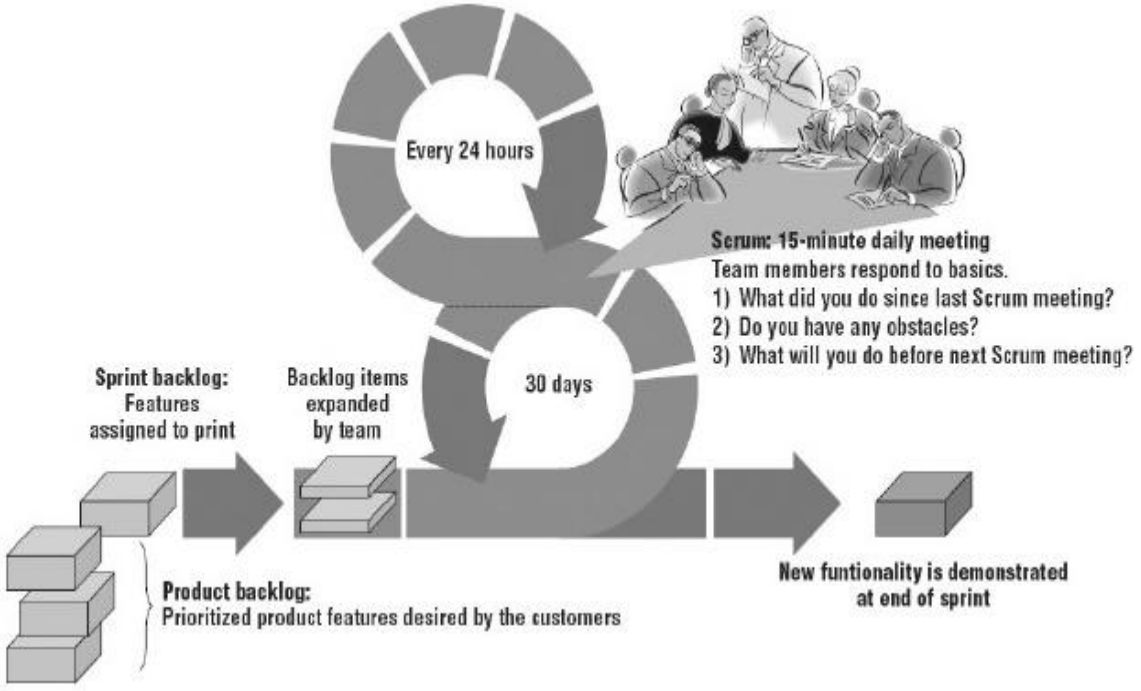

Figure 1: An example of agile software development methodology: Scrum

The Agile Manifesto articulates the common principles and beliefs underlying these methods (Cockburn, 2002), [21]. Among the first and perhaps best known agile methods are Scrum and XP (Salo, & Abrahamsson, 2008), [22]. See Figure 2 shows the current rate of Agile methodologies used. Scrum provides an agile approach for managing software projects while increasing the probability of successful development of software, whereas XP focuses more on the project level activities of implementing software. Both approaches, however, embody the central principles of agile software development [23].

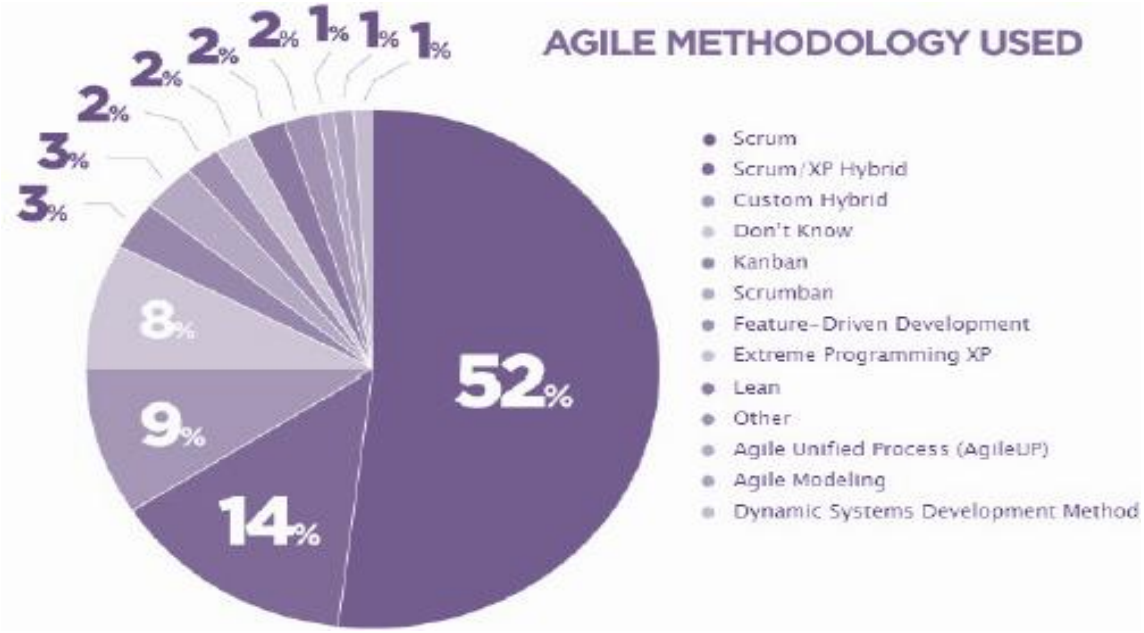

Figure 2: State of Agile Survey Results 2011 by Version One Inc. (Source: http://www.versionone.com)

Agile software development processes -- such as the Rational Unified Process (RUP), Extreme Programming (XP), Agile Unified Process (AUP), Scrum, Open Unified Process (OpenUP), and even Team Software Process (TSP) -- are all iterative and incremental (evolutionary) in nature [25]. Some these modern approaches, in particular XP and Scrum, are agile in nature. The agile methods are focused on different aspects of the software development life cycle. Some focus on the practices (extreme programming, pragmatic programming, agile modeling), while others focus on managing the software projects (the scrum approach) [24]

IV. BASIC ADAPTIVE AGILE METHODOLOGY METRICS

Three key views of Agile team metrics are typical of most implementations of Agile methods: velocity, sprint burn-down, and release burn-up. We present each of these with these cornerstones of Agile measurement. It is worth noting that metrics used in Agile development contexts tend to focus primarily on the needs of development teams, rather than on the status of a project or program.

A. Velocity

Simply stated, velocity is the volume of work accomplished in a specified period of time, by a given team. Typically, this is measured as story points accomplished per sprint. This measure is sometimes called "yesterday's weather" by Agile practitioners [Allerman 2009], as if to indicate its sensitivity to local conditions as well as seasonal trends. Indeed most experts explain that velocity is "team-unique" and thinking of this measure as a parameter in an estimating model is a mistake. The team must establish its own velocity for the work at hand [Coeldho 2009]. In addition, there are metrics associated with trends and stability of velocity—over time and/or across varying conditions. The height of each bar in figure below is the velocity for that sprint. Looking at this information, expecting this team to deliver 50 story points on the next sprint would seem unrealistic, since the team's previous highest velocity had been 35 story points in sprint 6. Conversely, a workload of just 15 story points would likely underutilize the talents of the team. For another team, or this same team working on a different product (or under different conditions), the expectations may be very different.

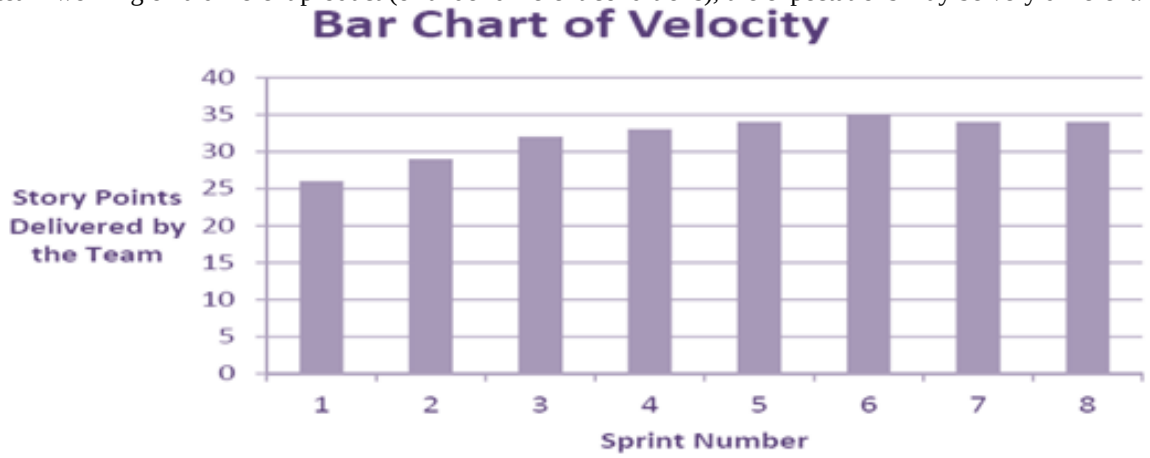

Figure 3: Velocity Chart

B. Sprint Burn-Down Chart

This graphical technique provides a means of displaying progress for the development team during a sprint. As items in the backlog of work are completed, the chart displays the rate and the amount of progress. This chart is typically provided for viewing on a team's common wall, or electronic dashboard. Many elaborations and alternative implementations are seen in practice, but, again – first the basics. Typically a line graph like the one in Figure 5, showing a decline in the remaining backlog items to be completed, is used. The workload chosen for that sprint (often called the sprint backlog) is reflected on the vertical axis—in story points for this example. A line sloping downward from left to right—with time depicted (in days) on the horizontal axis—shows the pace of work being completed. The dotted line shows the "ideal line" against which the thicker line can be compared.

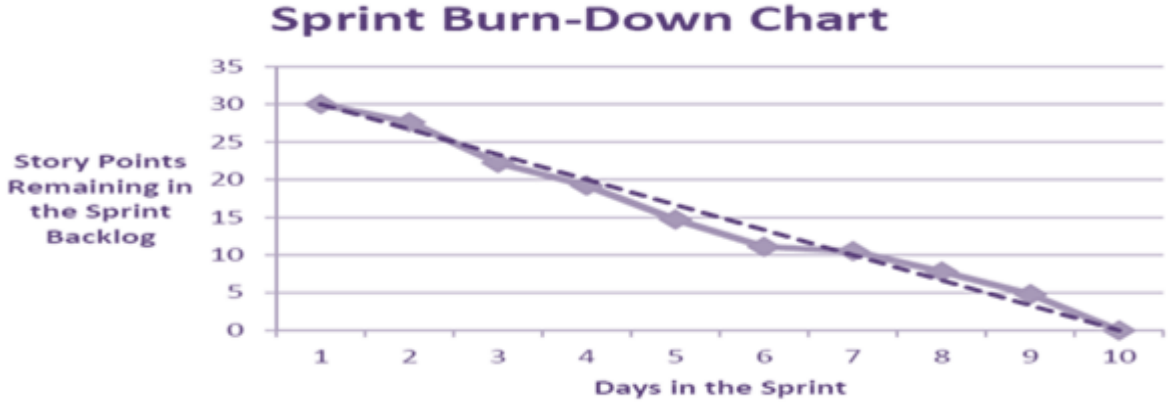

Figure 4: Sprint Burn- Down Chart

C. Release Burn-Up Chart

A complementary graphical technique for the sprint burn-down, the release burn-up chart is also commonly used. Many cling to the convention that sprints burn down and releases burn up— though there is no mathematical principle that governs this choice. With each completed sprint, the delivered functionality grows, and the release burn-up chart depicts this progress in an intuitively logical fashion as shown in Figure 5.

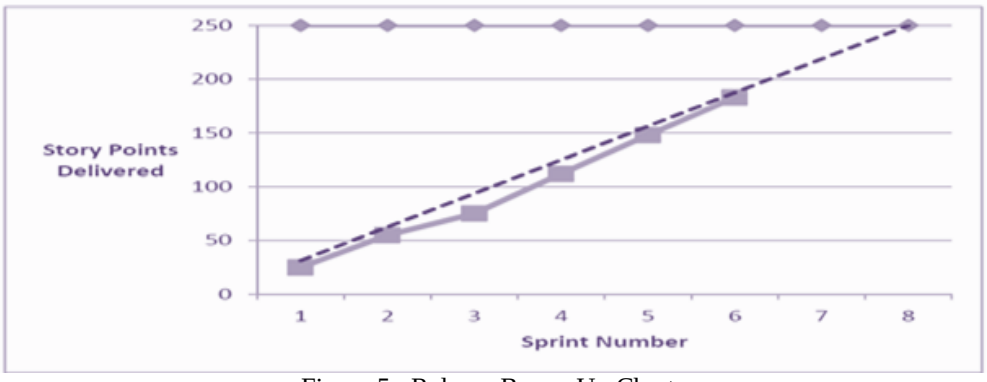

Figure 5 : Release Burn - Up Chart

V. SOFTWARE ARCHITECTURE QUALITY ATTRIBUTES AND UTILITY TREES USING ADAPTIVE AGILE

Common quality characteristics that can be evaluated at the level of architectural description are modifiability; performance and reliability. There are two aspects to consider in assessing the quality of a system's architecture

A. Architecture assessment

It allows us to measure the quality of a design. There are other complementary activities and methods for helping improve the quality of a system

1) *Systematic design processes*

Systematic design processes lead to high-quality designs. It helps the designer reason about the problem being solved and how best to solve it. The design process runs more effective since there is less solution searching. The process increases the number of potential solutions and thereby increases the chances of discovering a suitable solution. The quality of the requirements ultimately predetermines the quality of the system. The software architect often performs the role traditionally assumed by a systems analyst, that is, makes sure that the requirements are correct.

2) *Understanding the Right Problem*

Understanding the problem is a very important for quality of architectural design. It allows users to delete some data that was no longer required, but the system would not remove it permanently, rather mark it as hidden. It also allows users to change the state of some data so that it no longer affects the processing of the system but yet is still available for viewing and other operations. Al users can also extract data that is no longer needed in the operational database and place it into some archive. Allow users of the system to permanently delete unwanted data.

B. System Level View of Requirements

One of the most important things an architect can do is to analyze the dependencies among the requirements. This leads to a better decomposition of the problem.

C. Differentiating Design and Requirements

Whenever a feature is specified in a requirements document, there is a design (by human nature).It is important to be able to have a clear understanding of the problem even if elements of design are included. The architect must be skilled at analyzing requirements.

D. Assessing Software Architectures

Architecture assessment is the activity of measuring or analyzing a system's architecture in order to understand quality attributes. Architecture assessments should not only be performed by the architects, but by others also.

E. Reifying Nonfunctional Requirements

Most architecture assessment methods use scenarios as a way of simulating a system so that it may be reasoned about. Three categories of quality attribute:

1) External stimuli – events that cause the architecture to respond
2) Architectural decisions – design elements or design rules that have a direct impact on achieving attribute responses
3) Responses – how the system would react to the stimuli To make nonfunctional quality attributes measurable or observable they must be reified as concrete tasks or scenarios. Scenarios can be a very powerful specification technique because they make abstract requirements into tangible concrete tasks. Scenarios can be realized through the creation of a *quality attribute utility tree*, which forms the mains step of SEI's ATAM (Architecture Trade-off Analysis Method).

Example Quality Utility Tree

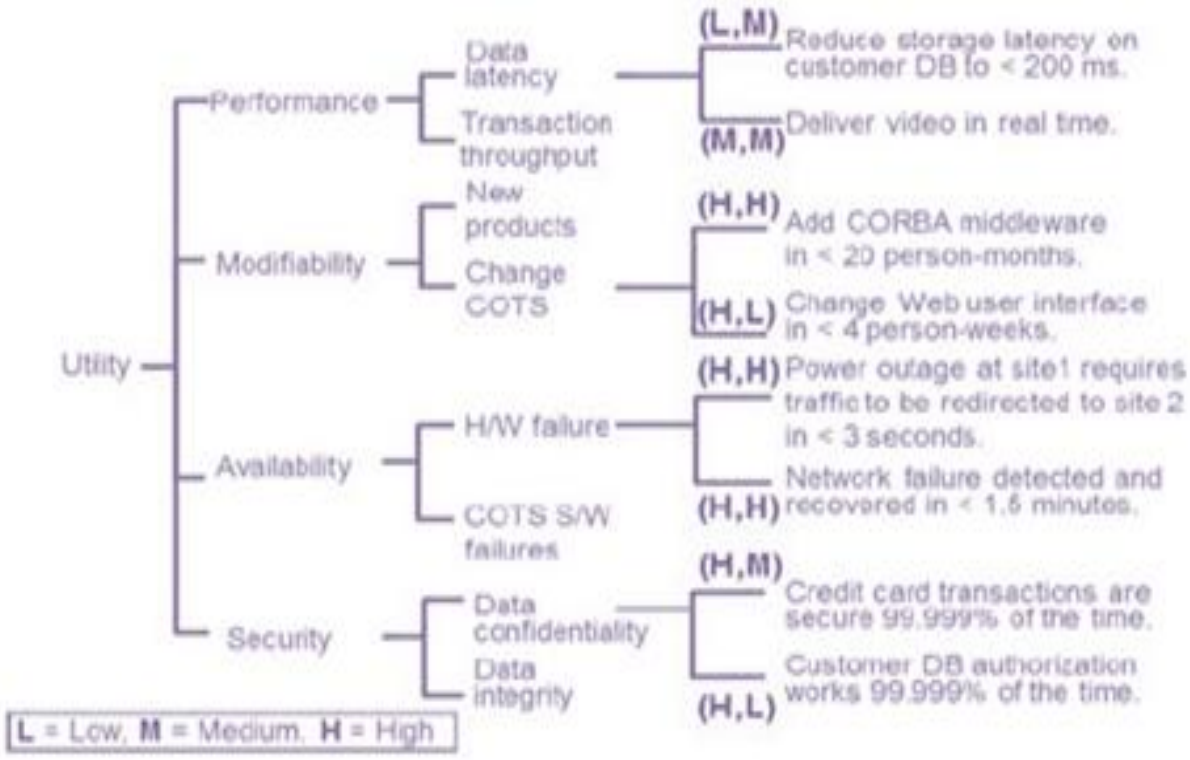

Figure 6: Quality Utility Tree

F. The Role of the Architectural Description

The architectural design models that comprise the architectural description can have a significant impact on the ability of an architect to analyze architecture. The architecture must be written down and it must use models that aid in its analysis.

G. Architecture Evaluation

Architecture evaluation is a development life-cycle activity in which several stakeholders analyze the architecture together in formal or informal ways using an assessment technique such as scenarios. Formal evaluations should be done as soon as the architecture is stable but before any commitment to development has been made.

H. Assessing Modifiability

Modifiability is a general quality attribute for specifying a change to a system. Modifiability requirements should be specified as specific scenarios (*change cases*).For example: Deploy a new product without recompiling or redeploying the runtime platform, replace repository with new vendor product and add an additional repository to the system.

I. Assessing Performance

There are two conflicting views with respect to performance: which includes "make-it-work-then-make-it-fast" and "filter-every-decision-through-a-performance-analysis. When considering architecture you can emphasize performance (this is really "make-it-work" in terms of the architecture). In detailed design you can concentrate on making it work first because design decisions are localized. One performance myth is that you have to have something to execute in order to test it for performance. Some aspects of performance can be predicted by analyzing the architecture. Many performance problems come from architectural design decisions. At the architectural level, performance is characterized by the interaction between components. By analyzing the communication characteristics between the components, performance can be predicted. Smith and Williams developed a software process engineering (SPE) method based on creating performance models for critical use case scenarios. The key scenarios are those normal scenarios that are performed frequently. A sequence diagram can be used to represent a use case scenario.

VI. AGILE ADOPTION

Agile methods are highly being adopted because of expectations that these methods can bring development success (Esfahani, Yu, & Annosi, 2010). One of the main reasons for success with agile methods is that they are highly adaptive (Boehm & Turner, 2003), [30]. Figure 7 demonstrates the current levels of agile adoption. In this case, 71% of respondents indicated that they work in organizations that have succeeded at agile and an additional 15% work in organizations that have tried agile but have not yet succeed at it.

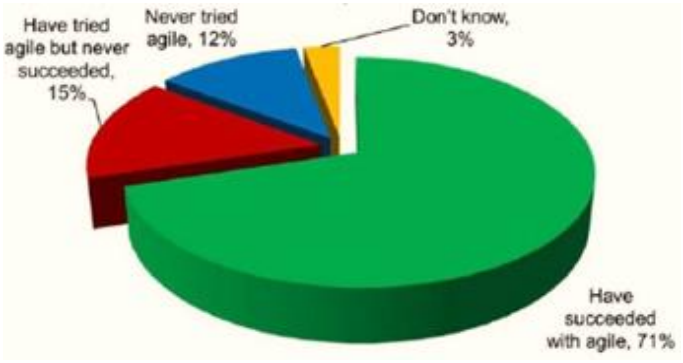

Figure 7: Agile Adoption Rate

Salo, O., & Abrahamsson, P. (2008), disagree that scientific publications and anecdotal evidence demonstrate that organizations worldwide are adopting agile software development methods at increasing speed [31]. In the study report of 2011, conducted by Forrester Research, the adoption rate of agile development approaches increases 35.4% to 38.6% whether as, traditional and iterative approaches declines. See figure 8.

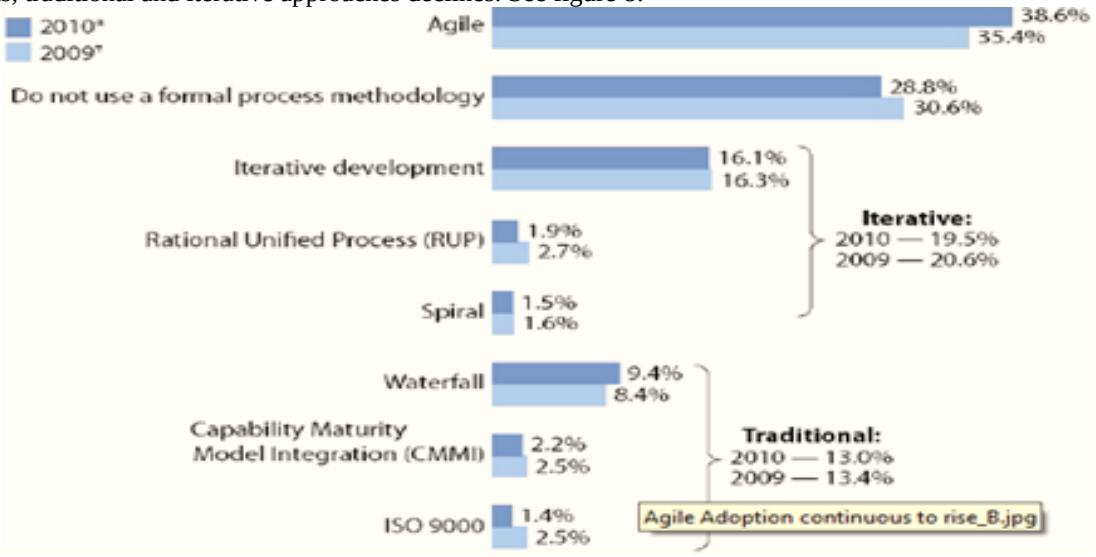

Figure 8: Forrester Research Agile Adoption Rate Rises (www.forrester.com)

According to (West & Grant, 2010), ― Agile processes have not only gained increasing adoption levels in the past few years; they have also rapidly joined the mainstream of development approaches‖ [32]. Mary large companies including HP, IBM,GM, Oracle, Intel and Microsoft use Agile methodologies [37] — and more and more smaller organizations turn Agile each year. In their study (West & Grant, 2010), conducted in 2009 by Forrester Research, 35% of organizations used agile software development processes, and another 16% of organizations used an iterative development approach, while only 13% of organization use a Waterfall approach. However, nearly 31% did not use a formal development methodology [22].

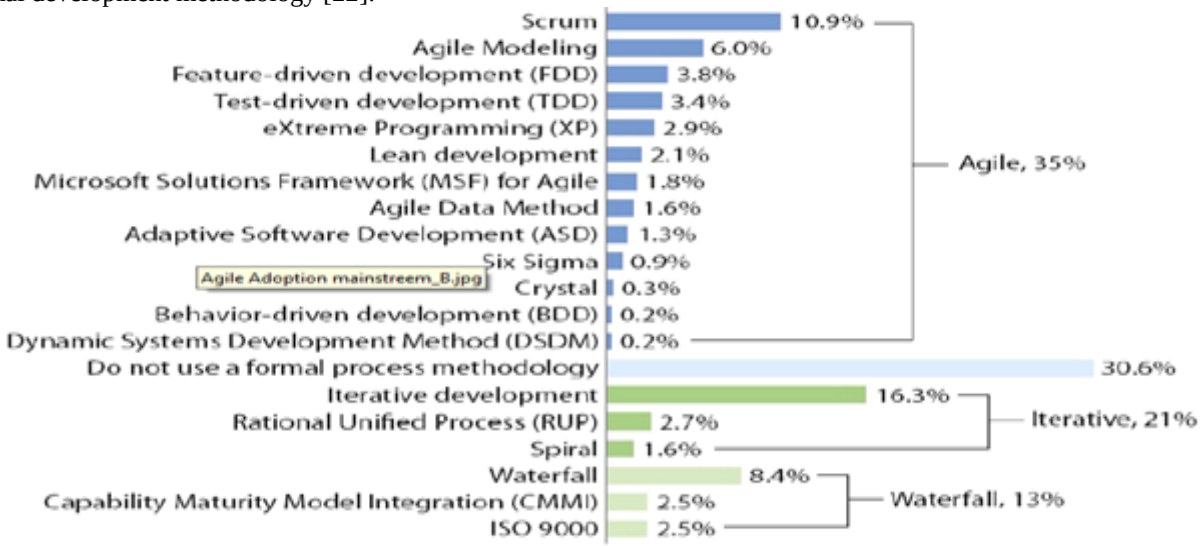

Figure 9 : Agile Adoptation Rates by Forrester Research in 2009

The main reasons behind for adopting Agile approaches rather than plan-driven approaches relate to: rapid changes; need for rapid results; emergent requirements (Boehm & Turner, 2003), [34]. According to Jason Charvat, (2003), D Leffingwell, (2007), & Perrin, (2008), Agile methodologies have many advantages including that they: adapt very well to change and dynamism; are people-oriented and value-driven, instead of process-oriented and plan-driven; mitigate risks by demonstrating values and functionalities up front in the development process; provide a faster time to market; improve productivity (by reducing the amount of documentation) and will fail early/quickly and painlessly, if a project is not doable [37], [36], [35]. A state of Agile survey was conducted in 2011 by versionone Inc. and the result shows: the top three reasons for adopting Agile to - increase productivity, accelerate time to market, and to more easily manage changing priorities.

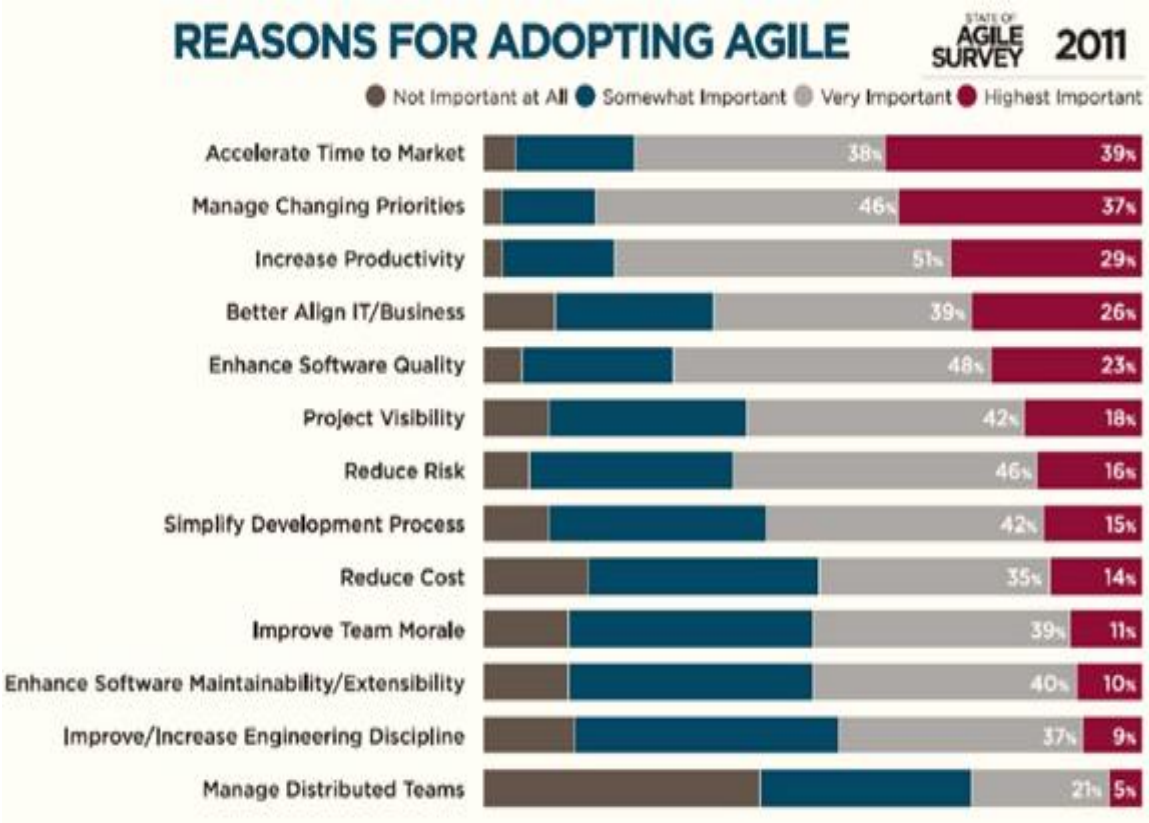

Figure 10: Reason for adopting Agile from "A State of Agile Survey in 2011" (www.versionone.com)

VII. CONCLUSION

This research has introduced a innovative tool for evaluation of agile methodologies against quality assurance factors. The quality management in agile methodologies is also new era of research. The limitation of tool is that is applicability in industry has not been verified. It may be due to difference in practices followed from organization to organization. The proposed tool would spark new era of research in field of agile software development especially the scenarios of agile modeling architecture for enterprise or offshore development. Architecture assessment allows the software development team to observe or measure the architecture of a system before it has been constructed in order to better predict the quality of the eventual system. This helps to lower development risks. Assessment can be formal or informal. It is necessary to know that the design has achieved an acceptable balance of quality attributes. The more complex a system is, the more important architectural design becomes. Architectural assessment gives us the knowledge needed in order to make the best architectural design decisions.

REFERENCES

[1] R. Harrison, '*Maintenance Giant Sleeps Undisturbed in Federal Data Centers'*, Computerworld, March 9, 1987.

[2] C. Hofmeister, R. Nord and D. Soni, *Applied Software Architecture,* Reading, MA: Addison Wesley Longman, 1999

[3] IEEE Architecture Working Group. *Recommended practice for architectural description*. Draft IEEE Standard P1471/D4.1, IEEE, December 1998.

[4] ISO/IEC, Information technology – Software product quality –Part 1: Quality model, ISO/IEC FDIS 9126-1:2000(E)

[5] I. Jacobson, M. Christerson, P. Jonsson and G. Övergaard, *Object-oriented software engineering. A use case approach*, Readings, MA: Addison Wesley, 1992.

[6] R. Kazman, G. Abowd, L. Bass and P. Clements, 'Scenario-Based Analysis of Software Architecture'. *IEEE Software*, 13 (6):47-56, 1996.

[7] R. Kazman, G. Abowd, L. Bass and P. Clements, 'Scenario-Based Analysis of Software Architecture'. *IEEE Software*, 13 (6):47-56, 1996.

[8] N. Lassing, D. Rijsenbrij and H. van Vliet. 'The goal of software architecture analysis: confidence building or risk assessment', In *Proceedings of the 1st Benelux conference on state-of-the-art of ICT architecture*, Amsterdam, Netherlands: Vrije Universiteit, 1999.

[9] P. Bengtsson and J. Bosch, 'Architecture Level Prediction of Software Maintenance', In *Proceedings of 3rd EuroMicro Conference on Maintenance and Reengineering(CSMR'99),* LosAlamitos, CA: IEEE CS Press, 1999, pp. 139-147.

[10] C. Larman, Agile and Iterative Development: A Manager's Guide. Boston: Addison Wesley, 2004. [84] C. Larman and V. Basili, "A History of Iterative and Incremental Development," IEEE Computer, vol. 36, no. 6, pp. 47-56, June 2003

[11] C. Larman and V. Basili, "A History of Iterative and Incremental Development," IEEE Computer, vol. 36, no. 6, pp. 47-56, June 2003

[12] V. R. Basili and A. J. Turner, "Iterative Enhancement: A Practical Technique for Software Development," IEEE Transactions on Software Engineering, vol. 1, no. 4, pp. 266 - 270, 1975. [86] B. Curtis, "Three Problems Overcome with Behavioral Models of the Software Development Process (Panel)," International Conference on Software Engineering, Pittsburgh, PA, 1989, pp. 398-399.

[13] B. Curtis, "Three Problems Overcome with Behavioral Models of the Software Development Process (Panel)," International Conference on Software Engineering, Pittsburgh, PA, 1989, pp. 398-399.

[14] Beck, K. (1999). Embracing change with extreme programming. *Computer*,*32*(10), 70-77.

[15] Schwaber, K., & Beedle, M. (2002). *Agile software development with Scrum*(Vol. 18). PTR Upper Saddle River^ eNJ NJ: Prentice Hall.

[16] http://en.wikipedia.org/wiki/Agile_software_development

[17] Stapleton, J. (1997). *DSDM, dynamic systems development method: the method in practice*. Addison-Wesley Professional.

[18] Highsmith, J. A. (2000). *Adaptive software development*. Dorset House.

[19] Cockburn, A. (2005). *Crystal clear: a human-powered methodology for small teams*. Addison-Wesley Professional.

[20] A Practical Guide to Feature Driven Development. 2002 S Palmer, M Felsing - Prentice Hall.

[21] Boehm, B. (2002). Get ready for agile methods, with care. *Computer*, *35*(1), 64-69.

[22] Pikkarainen, M., Haikara, J., Salo, O., Abrahamsson, P., & Still, J. (2008). The impact of agile practices on communication in software development. *Empirical Software Engineering*, *13*(3), 303-337.

[23] Salo, O., & Abrahamsson, P. (2008). Agile methods in European embedded software development organisations: a survey on the actual use and usefulness of Extreme Programming and Scrum. *Software, IET*, *2*(1), 58-64.

[24] http://www.ambysoft.com/essays/agileLifecycle.html#FigureDetailedLifecycle

[25] Turk, D., Robert, F., & Rumpe, B. (2005). Assumptions underlying agile software-development processes. *Journal of Database Management (JDM)*,*16*(4), 62-87.

[Allerman 2009]

[26] Allerman, Glen. *Deliverables Based Planning*, Denver PMI Symposium 2009.at http://www.slideshare.net/galleman/denver-pmi-symposium-2009

[Coelho 2012]

[26] Coelho, Evita & Basu, Anirban. "Effort Estimation in Agile Software Development using Story Points." *International Journal of Applied Information Systems (IJAIS) 3* (7): August 2012.

[29] Cohn, Mike. *User Stories Applied: For Agile Software Development*. Pearson Education, 2004.

[30] Boehm, B., & Turner, R. (2003). Using risk to balance agile and plan-driven methods. *Computer*, *36*(6), 57-66.

[31] Salo, O., & Abrahamsson, P. (2008). Agile methods in European embedded software development organisations: a survey on the actual use and usefulness of Extreme Programming and Scrum. *Software, IET*, *2*(1), 58-64.

[32] West, D., Grant, T., Gerush, M., & D'Silva, D. (2010). Agile development: Mainstream adoption has changed agility. *Forrester Research*.

[33] Scanlon-Thomas, E. (2011). *Breaking the Addiction to Process: An Introduction to Agile Project Management*. Itgp.

[34] Boehm, B., & Turner, R. (2003). Using risk to balance agile and plan-driven methods. *Computer*, *36*(6), 57-66.

[35] Lemétayer, J. (2010). identifying the critical factors in software development methodology FIT.

[36] Leffingwell, D. (2007). *Agile software requirements: Lean requirements practices for teams, programs, and the enterprise*. Addison-Wesley Professional.

[37] Charvat, J. (2003). Project management methodologies. *Selectingi, implementig, and supporting*.

[38] Hartman, Deborah & Dymond, Robin. "Appropriate Agile Measurement: Using Metrics and Diagnostics to Deliver Business Value." *Proceedings of the Agile Conference*. Minneapolis, MN, July 2006. IEEE Computer Society Press, 2006.

