AGILITY BY VIRTUE OF ARCHITECTURE MATURITY

Gouri Prakash
Business Risk Infrastructure Department, HSBC, U.S.A.

Keywords: Agile Software Process Models, Architecture Maturity.

Abstract: This position paper is to demonstrate how architecture maturity combined with the degree of agility observed in software projects, results in four different types of projects, namely Experimental, Conservative, Ceremonious and Optimizing and what the characteristics of each of these project types are. The paper puts forth the position that project managers, should endeavour to elicit characteristics of software projects and match them with the characteristics of the four project types, discussed in this paper, before embarking on the project. By pursuing this approach, managers would be able to ascertain the benefits realized in pursuing a particular project type for a given project and the difference between how things are and how things should be and what factors can get them to a should-be position.

1 INTRODUCTION

Proponents of conservative methods of software development that requires the observance of each phase of software development ceremoniously, in accordance with phases prescribed by the software process model have low levels of acceptance for agile methods, because agile methods accept the notion of changing requirements and adapt to the change. Agile practitioners on the other hand, are reverent of experiential knowledge and regard collaboration to be the guiding principle for current and future software development endeavours, being open to the idea of continuously learning while adapting as the software project progresses. They focus on adaptive development activities which can be reactive in nature, with little or no regard to the overall long term impact on the overall structural framework of the architecture as long as customer needs are meted on time. Agile software methodology hence facilitates the deployment of lightweight but disciplined methods of developing software in timescales shorter than those accomplished by the more traditional, conservative approaches. Is the traditional, conservative approach towards developing software a school of thought separate from that of agile software development or is it that there exist separate areas in the software development lifecycle (SDLC) process, that practitioners deliberately choose to focus on depending on the purpose and the context within which the software is developed. The move from traditional, conservative approach for software development to the more agile methods should be observed when there is sufficient evidence to support that the underlying system architecture is mature enough to facilitate agile software development. This is the recommended practice and the central theme of this poster. Agility, when observed in context of architecture maturity, becomes a metric that measures it. Mature architectures exemplify readiness for rapid deployment of software in a way that is useful for consumers of the software product.

2 ARCHITECTURE MATURITY

What is architecture maturity and what role does it play in enabling agile software development? Architecture maturity can be defined as the extent to which a given conceptual schema of software components serves as the representational design pattern for a software development endeavour, by virtue of past usage and performance of the schema in multiple endeavours of a similar nature. Architecture patterns that have a history of successfully realizing a software product, in multiple instances of software development projects, over a long period of time, are mature enough to be candidate architectures for existing and future projects and thereby facilitate and enable agile

Prakash G. (2010).
AGILITY BY VIRTUE OF ARCHITECTURE MATURITY.
In *Proceedings of the 12th International Conference on Enterprise Information Systems - Databases and Information Systems Integration*, pages 343-346
DOI: 10.5220/0002863703430346

software development methods. The architecture pattern has been used so often in a variety of projects that there is not much to be gained by trying to "reinvent the wheel" by focusing on the system architecture - instead the focus is on development and maintenance activities and being able to respond to changing requirements that meet the needs of business in an agile manner, because such a focus can be afforded, if the underlying architecture pattern is mature. For example, the N-tier architecture for developing an interactive website, that makes use of the J2EE framework for realizing the website implementation and that can be used repetitively in a software project with similar goals, is a mature architecture pattern.

The reason why agility is a function of architecture maturity is because a mature architecture reduces the technical risk associated with a given design pattern upfront, giving software practitioners more leeway in focusing on development and maintenance activities as opposed to architectural rightness as long as the experienced agile practitioners on the team are well-versed with the "ability" of the design pattern that is in use in the project, its application in the correct context and the extent to which the pattern can be stretched in existing and future software development endeavours. Additionally, from the standpoint of project management, mature architectures shorten the lead times required in delivering projects, thereby resulting in timely delivery of software projects. In short, mature architectures are "tried-and-tested" structural frameworks that facilitate agile software development.

2.1 Architecture Maturity and Agility Matrix

Presented next, in Fig 1, is the Architecture Maturity and Agility Matrix which shows the type of project that ensues based on the relationship between the degree of agility observed and the degree of maturity of the underlying architecture in a given software project.

Architecture maturity is on the X axis – and has two discrete states - either the underlying architecture for the software project is a "tried and tested" framework and hence exhibits high maturity or the underlying design pattern has not been around long enough to qualify as a mature architecture for software development projects and hence is categorized as one with low maturity. Agility, here on the Y axis represents the observance of agile software development methods in software projects and as such either agile methods are observed or not observed – hence "low" indicates lack of observance of agile development methods and "high" indicates that agile methods dominate the software development practices observed for the subject project. Based on the degree of agility observed in the software development project and the maturity of the underlying architecture pattern, the matrix in Fig 1, indicates the resulting project types followed by a description of these project types.

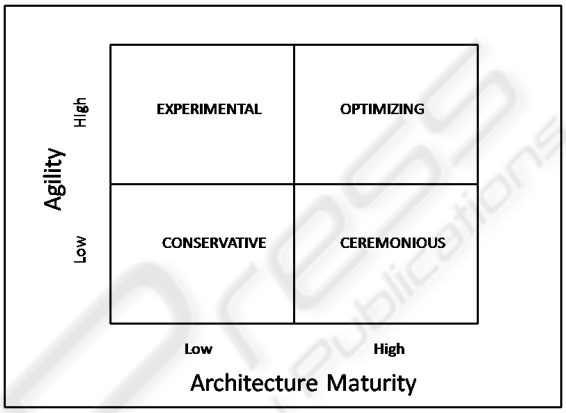

Figure 1: Architecture maturity and agility matrix.

2.1.1 Experimental

Software projects wherein the architecture maturity is low but usage of agile methods pervasive are primarily experimental projects wherein the team members are not afraid to try new techniques in order to quickly deliver on customer requirements. They are open to the idea of experimenting with a little known architecture pattern and choose to focus on developing workable software that can be delivered in short periods of time. For example, a company that wants to develop an intranet solution for its employees to keep them informed of the latest events happening in the organization can potentially adopt the Experimental model for doing the software project. The software team would develop the intranet using continuous feedback from their customers, in this case the employees, to further ascertain what content should be coded on the intranet and what features the customers find useful.

2.1.2 Conservative

Software projects that have low architecture maturity and where agile development methods are not observed, are primarily conservative solutions which take into account the high risk associated with a lesser known and little used architectural pattern and hence traditional methods for software

development are pursued for facilitating the software project. For example, a state government wants to install imaging systems at government controlled toll gates, that accurately capture graphic images of license plates of cars that pass through toll gates without paying the toll, which would then be used for generating letters sent to the person under whose name the car is registered with a statement of fines that the person of the car must pay to the government, in order to avoid charges of breaking the law purposefully. It is important that such a system have a near zero rate of false positives and false negatives and would have to be engineered using conservative approaches before being put in production. The project team implementing such a project should use conservative techniques given that the risks associated with system not working per the goal is unacceptable and that the architecture supporting the implementation has not been in use long enough to qualify as a mature design pattern.

2.1.3 Ceremonious

When the architecture maturity is high, i.e. the architecture is known to be robust in multiple instances of software development projects and when agile methods are conscientiously not employed and practitioners choose the traditional methods for software development, then the project observes a high degree of ceremony – a practice which can be revisited and re-assessed in the face of high architecture maturity to determine whether wastage of time and resources is occurring. Conservative projects can also be ceremonious and certainly vice versa, but conservatism in the former case is driven primarily by low risk tolerance associated with a little used design pattern and in the latter case can be more attitudinal and cultural however this may not necessarily always be the case. For example, with the Sarbanes Oxley Act of 2002, which applies to all organizations publicly traded on any one of the US stock exchanges, firms are expected to enforce detailed processes and extensive documentation for all software projects implemented within the organization and hence a culture of conducting software projects ceremoniously is the norm for these organizations.

2.1.4 Optimizing

Software development projects that make use of mature architectures and deploy agile software development methods are optimizing projects. These projects leverage architecture maturity to eliminate associated technical risks and make use of agile methodology to adapt to and respond to changing customer requirements. Software is developed incrementally on a sound architectural foundation with focus on implementing on-demand solutions in short periods of time. Optimizing projects are thus the result of high architecture maturity and high degree of agility observed during software development. Consider the case of context-aware IT solutions, such as PeopleSoft that are created to implement business processes relevant to a specific domain – that of payroll processing.

2.2 Characteristics of the Project Types

Having discussed the four quadrants of the architecture maturity and agility matrix, the question is - is one quadrant superior to others when it comes to deploying a project type that teams ought to engage in, given a software development endeavour? The answer is, it depends. Mature architectures were also once nascent structural frameworks which emerged borne out of the creative endeavours of practitioners and with use over a period of time proved to be mature enough to gain widespread adoption in endeavours of a similar nature. At the same time, different industries are subject to different government policies and often legal and compliance risks play a factor in choosing the more traditional, conservative and ceremonious approach to software development over agile methods. On the other hand, experimental projects can yield highly creative solutions and should not be discouraged if the team is aware of the risks they are taking when engaging in experimental software projects and the fact that there is a possibility that such projects can turn out to be of the “hit-or-miss” variety over the long term.

The optimizing solution does tend to leverage mature architectures – making use of agility in a calculated manner, thereby being able to deliver software in short cycles of time – a strong value proposition for the customer or end users of the software. It also leverages the maturity of the underlying architecture which plays a fundamental role in delivering reliable solutions. In fact, if the underlying architecture is known to exhibit a high degree of maturity, then agile methods should be considered the best practice for software development, unless there are other pressures or forces in play that leave the team with no choice but to adopt the ceremonious way of doing software projects. It is also worthwhile noting, that as the architecture maturity increases over a period of time, development teams potentially first engage in

conservative projects, then potentially shift to working on ceremonious projects and when the architecture is mature enough, opt to engage in optimizing projects. In the matrix, the move from conservative to experimental to optimizing is less likely than the move from conservative to ceremonious to optimizing.

Table 1: Project Types and their characteristics.

Characteristic	Experimental	Conservative	Ceremonious	Optimizing
Project Pace	Fast	Slow	Slow	Fast
Resilience to change	High	Low	Low	High
Risk Appetite of Stakeholders	High	Low	Low	Moderate
Customer involvement	High	Low to Moderate	Low to Moderate	High
Dominant Solution type	Workable	Facilitative	Formal	Improvi-sational
Technical Risk	High	High	Moderate	Low
Examples	SCRUM, XP	RUP, Waterfall	RUP, Waterfall	SCRUM, XP

The table above demonstrates the typical characteristics of project types given the degree of agility observed in performing the software project and the maturity of the architecture chosen for constructing the system.

3 CONCLUSIONS

Architectures that facilitate agility have the ability to satisfy customer needs and requirements more quickly and efficiently, but such architectures lie at the end of a spectrum of proven technological patterns. Emerging technologies and the consequent architectural patterns that implement these technologies as well as their widespread use and adoption contribute to making the architecture pattern mature which in turn creates a generation of "experienced" practitioners who advocate agility. There is scope for deploying agile methods for little known architectural patterns as well and this choice reflects the risks that the software development team and the stakeholders of the project are tolerant of when embarking on the software development projects. The objective of this poster is to facilitate conscientious decision-making when determining whether to adopt a conservative or ceremonious approach or an experimental or optimizing approach given the maturity of underlying architecture supporting the system under development.

The recommendation of this paper is for the project management team to initially make an evaluation related to the maturity of the architecture pattern utilized for the software solution, and an assessment of the level of risks involved and the amount of governance required to perform the project. Based on the evaluation and assessment a conscientious decision can be made on whether to use agile processes to implement the solution or traditional process models and in doing so implement a project type that is one of experimental, conservative, ceremonious or optimizing.

ACKNOWLEDGEMENTS

I would like to thank the sponsors of ICEIS 2010, for providing the opportunity to address and discuss issues and solutions related to software engineering.

REFERENCES

H. Erdogmus, "Architecture Meets Agility," *IEEE Software*, vol. 26, no. 5, pp. 2-4, Sep./Oct. 2009

K. Schwaber and M. Beedle, "Agile Software Development with Scrum", Prentice Hall, 2001

N. Rozanski and E. Woods, "Software Systems Architecture: Working with stakeholders Using Viewpoints and Perspectives", Addison-Wesley, 2005

Sidky, A. and Smith, G., *Becoming Agile in an imperfect World* , Manning Publications Co., Greenwich CT 2009

S. W. Ambler, "Agile Architecture: Strategies for scaling Agile Development", 2001-2008, http://www.agilemodeling.com

W. Cunningham, "Manifesto for Agile Software Development", http://www.agilemanifesto.com