The 8th International Conference on e-Business (iNCEB2009)
October 28th-30th, 2009

TAMING THE BUSINESS SYSTEMS DEVELOPMENT CRISIS WITH AGILE DEVELOPMENT METHODS

Roy Morien
roym@nu.ac.th
Department of Computer Science and Information Technology,
Faculty of Science, Naresuan University, Phitsanulok, Thailand

ABSTRACT

Traditional software systems development methods and project management methods, usually based on a civil engineering or construction engineering project management paradigm have been demonstrated as being inappropriate for the development of software systems.

This paper describes a development method, generally known as Agile Development, and Agile Project Management, and draws on what are considered to be reference disciplines and best practices from general management, and product development and manufacturing theory and practice, to provide guidance to the effectiveness of such methods. Given the ubiquity of software systems in organisations, adoption of this development approach is seen as an imperative for IT management to bring software development under control, to be more cost effective, and to provide greater business value to the organizations in which they are embedded.

Keywords: agile development, agile project management.

1. INTRODUCTION

From the earliest days of computer systems development a particular project management and development method paradigm has been adopted for the development of computer software systems. This paradigm was based fundamentally on the project management methods of civil engineering and construction engineering. Thus we have the terminology in software development of 'systems engineering'. This project management approach, and the associated development methodology was fundamentally a phased or serial approach comprising a significant requirements determination phase (Analysis) followed by an often lengthy design phase (Design) followed by a construction phase (Programming) and then a final Testing Phase prior to handover. Far too often the handover was accompanied by anger, frustration, disappointment and conflict. This approach was fundamentally adversarial, rather than collaborative and forming a development partnership, as the opposing parties sought to maximise their benefit, and minimise their cost.

This approach to systems development was favoured because it apparently provided an appropriate level of certainty at the outset; certainty of scope, certainty of cost and certainty of time. This desire for certainty was inevitably a vain quest.

This paper describes a development and project management method, now generally known as Agile Development, and Agile Project Management. This term includes Lean Development methods, and Lean Project Management methods. These approaches are considered to provide a better and more appropriate approach to systems development, and project management.

What may be called the reference disciplines of agile methods theory and practice will be investigated and discussed, to provide an intellectual basis for these methods. General management theory and practice is seen to provide a much better basis for software project management and systems development methods than the construction and civil engineering project management practices.

The reason for this is that software development is seen as a human activity, not a technical activity. Systems development activity is described as chaordic; a word that refers to a system that blends characteristics of chaos and order. Viewing systems development projects as chaordic activities explains the too frequent failure of the application of traditional engineering approaches to software development. This view can then lead us to consider management practices and theories other than engineering project management, and engineering management practices.

This paper considers these various other theory areas. Best practice manufacturing practices are discussed, and their applicability to software development is discussed. Kanban practices, The Model of Concurrent Perception, and its predecessor Model of Concurrent Engineering are canvassed as bases for agile development methods. Leadership studies, the Learning Organization, the post-industrial management theories are considered for their relevance and support of this model of 'soft' construction practices.

2. WHAT IS AGILE DEVELOPMENT

Drawing on a variety of sources [1, 2, 3, 4] and synthesising the definitions and suggested characteristics in those sources, Agile development is seen to have these characteristics:

- **People Focused:** (1) Collaborative: collaboration between developers and clients is continuous and continual, (2) Self-Organizing and Self-Managing Teams: Significant responsibility is handed to the team members, rather than a Project Manager, to decide on the work to be done in the next iteration.
- **Empirical and Adaptive:** Project management practices that have been published to support 'agile development' practices are described as 'empirical', 'adaptive', 'evolutionary' or 'experiential' rather than 'prescriptive', or 'pre-planned'.
- **Iterative:** Development is achieved through a series of short iterations each of which produces a useable enhancement to the system. This provides a frequent and regular feedback cycle, and opportunities for validation and verification of successful progress.
- **Incremental:** Development is achieved through a series of delivered increments to the system, each of which produces a fully developed, fully tested and certified extra feature or component of the system.
- **Evolutionary**: the system grows in size, the requirements *in detail* are continuously discovered, and are continually evolving during the development period.
- **Emergent**: the whole of the system is greater than the parts. The characteristics of the system emerge as parts are added.
- **Just-in-Time Requirements Elicitation:** Requirements are stated in detail 'just in time' to develop them, in the iteration in which those requirements will be implemented.
- **Knowledge-Based:** Development activity is decided upon by the knowledgeable, self-managing members of the team, with continual knowledge sharing about the product, the technology and the progress of the project. Learning and knowledge sharing are emphasized.
- **Client Driven,** 'Pull-Based' development: Only develop what is asked for by the Client, and when the Client asks for it.

Agile methods emphasize project transparency, continual communication and collaboration between project partners.

3. CHAORDIC SYSTEMS & ECOSYSTEMS

In the landmark book Birth of the Chaordic Age [5], Dee Hock coined the term 'chaordic' to describe the *"behavior of any self-governing organism, organization or system which harmoniously blends characteristics of order and chaos"*. It is suggested here that the activity of software development, as a system, manifests those characteristics. Latterly, the terminology of 'the digital ecosystem' has also indicated that *"The essence of digital ecosystems is creating value by making connections through collective intelligence. Digital Ecosystems promote collaboration instead of unbridled competition and ICT based catalyst effect in a number of domains to produce networked enriched communities."* It is suggested that a software development project is better described as an ecosystem, with many characteristics of being a digital ecosystem; *'promoting collaboration', 'a collaborative environment where species/agents form a ... coalition for a specific purpose or goal* (the development of a system)', *' creating value by making connections through collective intelligence'*.[6].

Therefore, it is suggested, software development projects, when defined in this way, defy a management 'command and control' style. As Champy [7] states, variously, *'You must have a culture that encourages qualities like relentless pursuit... bottomless resources of imagination ... and both smooth teamwork and individual autonomy'* (and therefore) *'... You cannot have a culture of obedience to chains of command and the job slot. It just won't work.'* And the best approach to such a system is *'... enabling (people); redesigning work so that people can exercise their skills and capabilities to the fullest extent possible – then stepping back and letting it happen.'* That is, let the ecosystem work, let the harmonious blend of order and chaos occur, and create a collaborative environment. The inappropriateness of hierarchical management structures, and 'command and control' style micromanagement is commented on in *Chaordic Organizations* [8] ... *'in a world that is ever more interconnected, it becomes more and more clear that no mechanical, top-down organizational structure based on control can be effective... the only effective organizations imaginable are those that are biological, guided by organizing principles, and that count on the full potential of people to think, to create, and to self-organize'*.

4. WHAT IS THE CUSTOMER BUYING?

Simply put, the Customer is buying a development activity. Traditional thinking seems to be that the Customer is buying a product. This is not the case! The Customer is buying the time, the expertise, the understanding, the experience and desire to provide a satisfactory outcome, of the Developers. The 'product' being purchased relies substantially on the willingness and ability of the participants to be creative, to collaborate, to learn and to pay heed to matters such as 'business value', 'quality', 'useability', 'usefulness', and so on. The Customer cannot, in any measure of reason, feel that their role is simply to pay the money, and receive the product, as if they are buying a new TV set at a department store. The Customer must participate, and the developers must welcome that participation. The developers cannot assume that they will be given a comprehensive, fully detailed, complete and unchanging specification of requirements that can be used as a given blueprint for the product they will construct. (Indeed, some Customers seem to put more thought into buying a TV set than they do in providing information about their system requirements).

The activity of software development is carried out over a period of time by various participants. It is a dynamic, often disorderly process. People do not behave in an orderly, serial, or linear manner. That is, it is a chaordic activity that requires an appropriate development method. Equally, it cannot be managed in the traditional way. The *'mechanical, top-down organizational structure based on control'* is not appropriate.

Fundamentally, the construction engineering project is about building or manufacturing a product. Given the necessary project planning and design, the builders know exactly what they are building. They have blueprints, plans, models, even videos and animations that allow the builders to view the 'finished' product. If this is not the case, then the project will be a 'disaster' if viewed through traditional eyes; for example, the Sydney Opera House that went a decade over time, and a hundred million dollars over budget (yet resulted in an acclaimed, architectural icon considered to be a contender for the 7 Wonders of the Modern World). In comparison, a software development project is product development, not manufacturing or building. It is almost impossible to create a blueprint of the proposed 'product'. The creativity and problem solving usually occur during the development, not as part of a settled blueprint at the start. Given this, it is again suggested that the adoption of the construction engineering or civil engineering project management approach was wrong, and inappropriate.

5. PROJECT SUCCESS CRITERIA

In the traditional Waterfall Approach, every attempt is made, at the start, to impose order and certainty on the process. Has this been successful? Can it be successful? Many research projects have identified the unfortunate statistics of failure, such as only 2% of systems were used as delivered, and 28% of systems that were paid for but were never delivered, and 47% of delivered systems were never used. *"...53% of projects overrun cost estimates by 189% or more (at a cost of US$59 billion per year in the US alone)"* [9] This research is supported by a US government study on software development projects, which revealed that 60% of projects were behind schedule and 50% were over cost (cited in [10]). The study also showed that 45% of delivered projects were unusable.

Some commentators have noted that there has in reality been a significant level of success in software development. Many thousands of systems have been developed within the broadest spectrum of complexity, without too many disasters being caused. This view is accepted as valid, but the fact is, many if not most systems have been developed within an environment of antagonism, misunderstanding, conflict, disappointment, and have provided far less business value than hoped for.

An important question is - Why is this? Perhaps it is because software project management activities are undertaken in this way in the mistaken belief that what sometimes works in independent manufacturing or civil engineering processes will succeed in software development. This view of software development as a manufacturing process, or an engineering process, has been significantly deprecated by many authors, even though it seems to have been a central philosophy upheld by many others – c.f.: the terminology of "software engineering". For example, *"...someone grabbed hold of the construction-manufacturing paradigm which suggests that we can layout an architecture, design the system, and construct it. Experience has shown that this is a painful and expensive way to develop dinosaurs."* and *"The construction paradigm is the major reason that so many customers are dissatisfied...."* and *'We need to shift from the old models of software development and maintenance – viz construction and manufacturing - to a new, more resourceful model of software development - software evolution'*. [11].

6. THE MODEL OF CONCURRENT PERCEPTION

Can there be an harmonious blend of order and chaos? Rubinstein et al [12] proffers a model of decision making behaviour that describes most compellingly the characteristics of chaordic systems. It is this model; *The Model of Concurrent Perception*, that seems entirely appropriate to the activity of software development. The Model of Concurrent Perception *'moves us from questions to answers, from divergent perceptions to convergent perceptions, from individual creativity to team implementation, from abstract thinking to concrete action, from quick experimentation to quality results, from deliberate chaos to emergent order'* and *'chaos should be deliberately created up front'*. By 'chaos' they mean that the situation be thrown open to participation and discussion by all interested stakeholders, and a rich mix of views, opinions, suggestions, expertise and ideas be aroused, thus creating a 'chaotic' situation from which order will emerge. *'Questions need to be raised from the outset. When you start out with divergent questions, you will end up with convergent answers. When you start out with chaos, you will end up with order.'*

To create 'certainty' at the start of a project is to imply that the future can be controlled, which is a fallacy. Uncertainty is the hallmark of the future. Ours is not to know the future, but just to plan for it, including whatever contingencies can be foreseen. The Model of Concurrent Perception says this in this way; *'This* (start with chaos, end up with order) *is far preferable to the scenario where everyone coasts through a seemingly structured and orderly project and the end result is chaos'*. This last phrase seems to almost perfectly describe the traditional phased software development approaches where every effort is made, by the creation of a detailed and 'frozen' requirements specification, and a detailed plan that is rigorously held to, to have *'a structured and orderly project'*. 'Plan the work and work the Plan' is seemingly the motto of the 'successful' project manager. Research has shown that the end result of such an approach seems too often to end in chaos, characterized by disappointment, rejection and refusal to use the resultant system. The statistics about project outcomes success referred to previously clearly indicate a descent into chaos from a starting point of imposed order and directed certainty. Assuming that these systems were developed using a traditional phased approach, which is not an unreasonable assumption, we can see relevance and correctness of the situation of '*seemingly structured and orderly project*' where the '*end result is chaos*'.

There are examples of highly successful projects from other than software development that seem to be well described by the Model of Concurrent Perception (and its predecessor Model of Concurrent Engineering) including the development by the Boeing Corporation of the 777 airliner, and the development of the Lexus luxury motor vehicle by the Toyota Company.

In the development of the 777 airliner, for example, the project manager '*created more than 200 design/build teams with members from design, manufacturing, suppliers and customer airlines – everyone from pilots to baggage handlers*'. [13]. All project teams and members were urged to '*share early and share often*'. The project scenario being painted here is clearly the '*...start out with chaos*' situation, which, in this case, resulted in the creation of a highly successful airliner which is clearly the manifestation of '*you will end up with order*' theory of the Model of Concurrent Perception.

In the development of the Lexus motor vehicle, as described in Liker [14] it is stated that '(in vehicle design) *Effectiveness starts with what is popularly being called the 'fuzzy front-end*''. The project leader stated '*The end result was not just my effort alone, but all the people along the way who originally opposed what I was doing, and who all came around and were able to achieve all these targets that I had set in the first place*'. The Lexus motor vehicle is known as a very popular model in the marketplace. Various aspects of the Model of Concurrent Perception were clearly able to be seen here. '*Questions need to be raised from the outset*'; indeed many questions were raised about design issues, even about the need for the model. '*When you start out with divergent questions, you will end up with convergent answers*' was demonstrated by the people '*who all came around*' and achieved the design targets. '*When you start out with chaos, you will end up with order*'.

7. THE LEARNING ORGANIZATION

Elsewhere we can go to the literature about a management discipline outside IS and Computer Science to seek insight into the best way to develop software systems. In this case, to view the software development function in terms of being a Learning Organization.

The concept and practice of the learning organization is amply discussed in Senge, in his book entitled '*The Fifth Discipline-The Art & Practice of the Learning Organization*'[15]. Peter Drucker defined a learning organization being necessary because '*The function of the society of post-capitalist organisations ... is to put knowledge to work ... it must be organised for constant change*'.

The Core Capabilities of a Learning Organization are summarized as (1) Creative orientation, (2) Generative discussion, and (3) Systems perspective (Maani & Cavana, [16] at p138.). These concepts are elaborated to mean:

- Creative orientation: The source of a genuine desire to excel. .. The source of an intrinsic motivation and drive to achieve ... favors the common good over personal gains.
- Generative discussion: A deep and meaningful dialogue to create unity of thought and action
- Systems perspective: The ability to see things holistically by understanding the connectedness between parts.

Although Senge published nine years before Rubinstein & Firstenberg ([12, op.cit.] it is interesting to see the many similarities between their discussion. In discussing Team Learning, Senge states (at p.236) '*team learning (has) the need to think insightfully about complex issues ... to tap the potential of many minds*'. Other statements about team learning include '*... team learning involves mastering the practices of dialogue and discussion ... there is a free and creative exploration of complex and subtle issues ...* '.

This implies, it is suggested, the chaos that is present in the participant behaviour modelled by the Model of Concurrent Perception, and then the learning team converges on the order that is the hoped for outcome of '*divergent perceptions to convergent perceptions*'.

Similarly, when Maani & Cavana [16, op.cit.] refer to '*Generative discussion: A deep and meaningful dialogue to create unity of thought and action*', we can reasonably interpret the '*deep and meaningful dialogue*' to be the chaos and the '*create unity of thought and action*' to be the emergence of order, all of which seems readily defined by the Model of Concurrent Perception.

An interesting if less than proven concept is known as 'the wisdom of the herd', or a little more eloquently as 'collective wisdom' [17]. Applying this to agile methods, we see the efficacy of team self-organization, where planning and development decisions are made by the collective team, not by a commanding and controlling project manager.

8. LEAN PRODUCT DEVELOPMENT

Lean production was a manufacturing methodology developed originally by the Toyota Motor Company. It is also known as the Toyota Production System. The goal of lean production and lean product development is stated as 'to get the right things to the right place at the right time, the first time, while minimizing waste and being open to change'.

The Toyota Production System was masterminded by Taiicho Ohno who is credited with developing the principles of lean production, discovered that in addition to eliminating waste, his methodology led to improved product flow and better quality.

The overall management philosophy and practice of Toyota is stated as 14 management principles in Liker [14, op.cit.]. Instead of devoting resources to planning what might be required for future manufacturing, Toyota focused on reducing system response time so that the production system was capable of immediately changing and adapting to market demands. This became known as a Kanban, and otherwise as Lean Product Development, under the general term The Toyota Production System. The principles of lean production enabled the company to deliver on demand, minimize inventory, maximize the use of multi-skilled employees, flatten the management structure and focus resources where they were needed. Ten rules of lean production were stated. These were:

1. Eliminate waste
2. Minimize inventory
3. Maximize flow
4. Pull production from customer demand
5. Meet customer requirements
6. Do it right the first time
7. Empower workers
8. Design for rapid changeover
9. Partner with suppliers
10. Create a culture of continuous improvement

Techtarget Network [18]

Summarising the 'success attributes' of the Toyota Production System:

- Fostering an atmosphere of continuous learning and improvement
- Satisfying customers (and eliminating waste)
- Quality first and consistently
- Grooming leaders from within the organization
- Teaching employees to become problem solvers
- Growing together with suppliers and partners for mutual benefit.

9. LEAN DEVELOPMENT IN SOFTWARE PROJECTS

These principles have been applied to software project management in this manner: [13, op.cit. 19]

In Manufacturing	In software development
Reduces Inventory	No unnecessary and potentially obsolete and invalid documentation, or code
Improves Flow	Surfaces impediments in the development process quickly, including bottlenecks, waiting etc.
Prevents Overproduction	No code is developed that is not demanded by the client that may be unusable and not useful
Places control at the operational level	Self Managed teams empowers developers and enhances creativity
Creates visual scheduling and management of the process	Transparency in the development process and up-to-date information available to all.
Improves responsiveness to change in demand	Satisfies contemporary client requirements rather than potentially obsolete requirements
Minimizes risk of inventory obsolescence	Developed code is always needed, and does not become obsolete and unnecessary.

10. LEADERSHIP AS A WINNING STRATEGY

In 1995, a team from New Zealand won the famous and prestigious yacht trophy, called the Americas Cup [16, op.cit.). This was only the 2nd time in 146 years that a non-US syndicate had won the trophy ... Australia had won it once before.

The amazing thing was that the NZ yacht won 41 of the 42 races that they competed in over the 6 months competitive campaign. What was even more amazing was that the NZ syndicate had a very limited budget, and a limited amount of time to develop their record-winning boat.

How did they do it? There are some valuable lessons here in this experience and success that are very applicable to software development. Indeed, a significant contribution to the success of this syndicate was the evolutionary development of a computer system of sailing models that assisted in preparing the boat for competition. The success has been attributed to:

- The inspirational leadership of the syndicate Leader
- The strong sense of community within the syndicate team
- The openness of communication between team members
- 'Customer'- led development – the sailors!!!
- The sustained rate of continual improvement (of the boat speed)
- The level of commitment and purpose by all participants

This syndicate exhibited many of the valuable traits of a 'learning organization'. The contribution of 'leadership' to this outstanding success also cannot be underestimated. Without the 'inspirational leadership' of the syndicate financier and leader, all else may well have been in vain. The 'vision' was developed and pursued by the syndicate leader, providing the authority to proceed, with the team behaving in a highly collaborative and self-managed manner, iteratively (that is, from race to race) providing incremental improvements to performance based on an 'inspect and adapt' philosophy, wherein the system 'evolved'.

11. KANBAN – THE PRACTICE OF SIGNS AND JUST-IN-TIME

Drawing from a leading book on the topic of Kanban [19] the introductory chapters about Kanban are paraphrased here, and the text is applied to software development.

Kanban is about scheduling for manufacturing. The purpose of Kanban is to ensure the availability of supplies to the production line 'just in time' when those parts are needed. Kanban is a small batch oriented system of supply chain, based on a 'pull' or demand driven supply of product. Kanban uses a simple system of signs to indicate the need for another batch of inputs.

These are the four main and essential characteristics of Kanban; that is, Just-in-Time scheduling, small batch oriented, demand driven or 'pull' driven demand, and simple radiating of information by a simple system of signs. Kanban essentially eschews elaborate and complex MRP systems and detailed planning, in favor of simplicity. Kanban was pioneered by the Toyota Motor Company and was part of the system of manufacturing, now known as Lean Manufacturing that brought Toyota to the position of being the world's leading and most profitable motor vehicle manufacturing company.

The production process using Kanban controls produces product only to replace the product consumed by customers; that is, demand driven or pull-driven. Applying this to the software development process, we can say that a 'kanban' approach to software development means that only software with characteristics requested by the client will be developed. This aspect of 'software kanban' eradicates what is known as gold-plating, where developers include apparently sophisticated 'bells and whistles' that may only be interesting to the developer, which have not been requested by the client, and may never be used. Research has indicated that up to 60% of features in a software package are never used by the client. It is therefore a waste of time, money and effort to include them.

The 'Just-in-Time' supply philosophy of Kanban in manufacturing has resulted in substantial cost savings in the maintenance of on-hand inventories, and the management of those inventories. Significant wastage can occur if existing inventories become unusable because of changing demand patterns. Applying this philosophy to software development can see a substantial change in the development process, which, in the traditional manner, produces huge quantities of 'product'; in this case requirements documentation, as an example, at the start of the development process, and this inventory of product becomes obsolete quickly, given the inevitable change in requirements brought about by two major factors (as discussed previously); that is, it is impossible to fully and comprehensively detail all requirements at the start of the project, and the equally inevitable fact that requirements will change during the development process, as more is realized, new ideas created, and the shortcomings of the initial requirements documents understood. Another inventory item in the software development process is unrequested code, created 'just in case' the client requests it, or 'just in case' it might be useful in the future. Kanban philosophy and practice suggests that it would be far better to replace 'just in case' development with 'just in time' development.

Kanban, in the manufacturing process, replaces the daily scheduling activities necessary to operate the production process, and thereby removes the need for substantial production planning staff and continuous monitoring of the production process. This places control at the 'value-added' level of production and empowers the operators to control the line. Applying this to the 'agile' development process of software development, the 'self-managing team' approach to software development, this creates a more creative and productive development environment.

In agile development, the insistence on highly visible 'information radiators', and the essential transparency of project 'progress' is equivalent to the Kanban use of visible but simple signs and signals that are obvious to everyone, and that flag progress and demand in a transparent and effective manner.

The highly iterative style of agile methods, where work is planned and done and delivered in periods as short as one week, is emulating the Kanban practice of small batches. Frequent adoption of a 'small batch' of work to be achieved in a short period of time is seen as a highly successful approach.

The benefits of Kanban scheduling, as tabled in this book, include [20]:

- **Physical:** It is a physical card. It can be held in the hand, moved, and put into or onto something.
- **Limits WIP:** It limits WIP (Work-In-Process), i.e. prevents overproduction.
- **Continuous Flow:** It notifies needs of production before the store runs out of stock.
- **Pull:** The downstream process pulls items from the upstream process.
- **Self-Directing:** It has all information on what to do and makes production autonomous in a non-centralized manner and without micro-management.
- **Visual:** It is stacked or posted to show the current status and progress, visually.
- **Signal:** Its visual status signals the next withdrawal or production actions.
- **Kaizen:** Visual process flow informs and stimulates Kaizen.

Whilst warning against attempts to apply all of the aspects of a manufacturing control process to software development, in this view of what the author terms Agile Kanban, the focus is more on enabling tasks, "Visual" and "Self-directing," so as to help the team become autonomous and improve their own process. In order to make the process continuously flow as well as to limit WIP, "iteration meetings" are needed to communicate the information. The use of the Kanban process control approach has been proven as being highly effective in improving process efficiency and efficiency in manufacturing. It is possible to apply at least some of these principles and practices to software development, and applying its principles to software development provides a significant support to the use of Agile methods in software development, supporting the assertion that agile methods are an effective and efficient way to develop software systems, reducing cost and time to market, improving quality and business value, and creating a significantly greater level of user satisfaction with the outcome.

12. CONCLUSION

It is suggested that the traditional software development methodological and project management paradigms based on civil engineering and construction engineering practices are deficient and inappropriate. A different paradigm that acknowledges that software system development activities are essentially 'chaordic', and require leadership and behavior styles appropriate to this type of project ecosystem, has been proffered; Agile Development.

Support for the appropriateness and efficacy of Agile Development has been drawn from a variety of sources and reference disciplines, drawing on the experience of highly successful companies and ventures, and the theories and practices of management science, such as Leadership studies, The Learning Organization, Kanban, The Model of Concurrent Perception, Lean Manufacturing and Lean Processes and so on. Taken together, and drawing on the characteristics of these various areas of theory and practice suggests strong support for the proposition that Agile Methods are very well based on leading management theories and practices, and are indeed a highly effective and efficient way to develop software systems. Given the ubiquity of computer systems in today's world, and the almost existential reliance on computer systems in every operational organization, the proposition is stated that adopting and implementing Agile methods of software development, and software project management, is an imperative and can be adopted with confidence and impunity by modern organizations.

REFERENCES & BIBLIOGRAPHY

[1] Agile Alliance http://www.agilealliance.org, accessed July 30th, 2009

[2] Agile Manifesto, http://agilemanifesto.org/, Accessed July 30th, 2009

[3] Mahanti, A. (2006). *'Challenges in enterprise adoption of agile methods - A survey'.* Journal of Computing and Information Technology **14**(3): 197-206.

[4] Melnik, G. and F. Maurer (2005). *'Agile methods: A cross-program investigation of student's perceptions of agile methods'.* ICSE'05, ACM Press, IEEE.

[5] Hock, Dee, *Birth of the Chaordic Age,* Visa International, 1999

[6] http://www.business.curtin.edu.au/business/research/debii-tier-1-institute accessed July 30, 2009

[7] Champy, James, Reengineering management: The Mandate for New Leadership, Harper-Collins Publishers, 1995

[8] *'Chaordic Organizations',* www.paricenter.com/library/papers/**Chaordic**_organizations.pdf, *accessed July 20, 2009*

[9] Standish Group (1994). *The Chaos Report (1994)* [online]. Available WWW: http://www.standishgroup.com/sample_research/chaos_1994_1.php Accessed December 14th, 2003

[10] Garmus, David and David Herron (2001), *Estimating Software Earlier and More Accurately*, excerpted from Function Point Analysis Measurement Practices for Successful Software Projects, Addison-Wesley Information Technology Series, 2001.

[11] Arthur, L.J., *Rapid Evolutionary Development: Requirements, Prototyping & Software Creation*, Wiley, 1992

[12] Rubinstein, Moshe F. and Iris R Firstenberg, *The Minding Organisation*, John Wiley & Sons, 1999.

[13] Mary,Tom Poppendieck, *Implementing Lean Software Development: From Concept to Cash,* Pearson Education, 2007

[14] Liker, Jeffrey K., *The Toyota Way*, McGraw-Hill, 2004

[15] Senge, Peter, *The Fifth Discipline – The Art & Practice of the Learning Organization,*

[16] Maani, Kambiz E. & Robert Y. Cavana, Systems Thinking, Systems Dynamics – Managing Change and Complexity, Pearson Education NZ, 2007

[17] 'the wisdom of the herd', http://en.wikipedia.org/wiki/Collective_wisdom, accessed July 30th, 2009

[18] (Techtarget Network, 2005) http://searchcio.techtarget.com/sDefinition/0,,sid19_gci810519,00.html

[19] Gross, John M. and Kenneth R. Mcinnis, *Kanban Made Simple*, American Management Association, 2003.

[20] http://www.infoq.com/articles/hiranabe-lean-agile-kanban#content, accessed July 15, 2009