Int. J. Advance Soft Compu. Appl, Vol. 6, No. 1, March 2014
ISSN 2074-8523;

An Agile Software Project Management Manifesto – A Reference Disciplines Framework for Agile Development

Roy Morien

roym@nu.ac.th

Editor, Naresuan University Language Centre, Phitsanulok, Thailand

One-time Research Fellow, Curtin University, Western Australia

and Visiting Specialist, Department of Computer Science and Information Technology, Naresuan University, Phitsanulok, Thailand ,Sometime Visiting Lecturer, Kasetsart University, Bangkok, and Silpakorn University, Nakhon Pathom, Thailand.

Abstract

The Agile Development approach to systems development is fundamentally based on the Agile Manifesto published in 2001 by a group of independent consultants and practitioners. Twelve Principles supporting the Agile Manifesto were elaborated in support. Following the spirit of the Agile Manifesto, this paper proffers the Agile Project Management Manifesto. In part, this Agile Project Management Manifesto looks towards identifying and proposing appropriate reference disciplines which are manifest in the Agile Development Universe of Discourse. The paper will attempt to bring together concepts and practices of 'agile development', 'lean product development', 'chaordic systems', 'leadership studies' and concepts of the 'learning organisation, together with 'the Model of Concurrent Perception' to suggest a new Paradigm of Software Development and Project Management. This new paradigm is framed by the Agile Project Management Manifesto described. The Manifesto evokes some reference disciplines to support its various aspects. Why is it desirable and necessary to search for such reference disciplines. The answer is that Alternative Software Development and Project Management approaches, such as Software Prototyping and Rapid Development. and now including Agile Development, all seemed based on pragmatic experience and have been designed mostly by consultants and practitioners. While this is not in any way to denigrate or deprecate them, it is felt that support for these approaches needs to also come from other disciplines, such as management studies, leadership studies and the like to give Agile Development a more solid theoretical and intellectual basis.

Keywords: *Agile Development, Agile Project Management, Software Development*

1 Introduction

In 2001 what is known as the Agile Development Manifesto was published by a group of consultants and practitioners. The Manifesto clearly was based on the extensive experience and knowledge of software development practice and processes. The dimensions of the Agile Development Manifesto, and the more explanatory Twelve Principles supporting the Manifesto were clearly counter-intuitive to the then widely accepted Waterfall Approach to software development. The thinking behind the Agile Manifesto clearly attempted to identify and differently define the true nature of the software development process. Such definition of its characteristics is missing from the literature. The names in use clearly imply an activity akin to civil engineering and construction projects. Principle among these is the set of practices termed Software Engineering. In Object Oriented development, the Pattern oriented approach includes a Pattern of 'software factory'. We see discussion about the role of the Database Architect, a primarily construction role and activity. Project planners are exhorted to 'plan the work and work the plan', and requirements documents are referred to as blueprints. A professional project management activity is the development of a Gantt chart, and students of Project Management inevitably are taught to use some form of project management tool, such as Microsoft Project(R).

The Waterfall Model of development, first described by Royce [1] clearly seeks to achieve certainty in the future project activities and certainty in outcomes. The Waterfall Model was originally formalised by authors including Tom deMarco [2], Edward Yourdon and Larry Constantine [3] and Chris Gane and Trish Sarson [4] under the name of Structured Design, or Structured Analysis and Design. It is in common use still, even though these principles were published in the mid- to late-1970's. This was during a time when most system being developed could be described as 'batch systems replacing manual clerical systems'.

This paper will explore an alternative view of software development projects, and propose a new way of thinking about the activity of software development, and software project management. The discussion will draw upon management literature normally considered outside the scope of software project practice, and will explore the nature of the software development activity, and consider a new approach to software project management based upon the different characteristics identified as being the more correct nature of that activity, not the engineering oriented nature that has hitherto been considered the received way.

2 The Agile Project Management Manifesto

The original Agile Development Manifesto [5] as published by a group of consultants and developers as the fundamental basis for a new way to develop software, is shown in Figure 1.

We are uncovering better ways of developing software by doing it and helping others to do it. Through this work we have come to value:		
Individuals and Interactions	over	Processes and Tools
Working Software	over	Comprehensive Documentation
Customer Collaboration	over	Contract Negotiation
Responding to Change	over	Following a Plan
That is, while we value the items on the right, we value the items on the left more.		

Figure 1: The Manifesto for Agile Software Development

In the same manner of thinking and publication, this author has developed what is termed a Manifesto for Software Project Management, which attempts to encapsulate the philosophy of Agile Development applicable to the management of software projects, and as a framework for the reference disciplines considered relevant to agile project management. This is an original concept published here for the first time.

We are seeking better ways of managing software development projects. Through this endeavour we have come to value:		
Leadership	not	Management
Team Empowerment	not	Compliance to Control
Collaboration and Cooperation	not	Organisation Hierarchy
Adaptive Behaviour	not	Planning and Estimating Rigidity
Experiential Learning & Development	not	BDUF and Static Requirements
Welcome Change	not	'Frozen Specs' and fear of 'scope creep'
That is, while traditional thinking values the items on the right, we value the items on the left instead.		

Figure 2: The Manifesto for Agile Software Project Management

Support for these concepts has been found in the literature, but, unfortunately, not in the Computer Science or Information Technology literature (apart from inference and hint in the published literature on Agile Development.

The remainder of this paper will discuss the variety of reference disciplines considered necessary and appropriate to support the Agile Software Project Management Manifesto.

3 The Three Processes in Software System Development

Development of a system should not, and cannot, be seen as a single process. There are three interacting and contributing processes in the development of a computer system. But first it must be recognised and acknowledged that the system will be embedded in the organisation, and the primary purpose of the system is to support and enhance the activities and management of the organisation in which it is embedded.

The three systems (or activities) referred to are illustrated in Figure 3. :

This diagram suggests the nexus between the three activities. The Business Activity is the given basis for a required system, and the Computer System must be solidly based on the requirements of the Business Activity. Depending on the style of the Computer System Activity, an appropriate System Development approach can be decided upon. It is acknowledged that an Internet based system, for example, will have different development requirements to perhaps an on-line interactive database processing system, and if it is cloud-based will require different skills and development activity than otherwise.

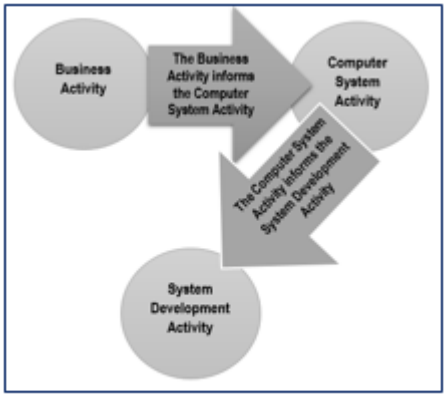

Figure 3: The Three Activities

It is clear then, that there must be collaboration between the participants in the Business System who will be users of the computer system, and the developers who must be adaptable and more than just technically minded, depending on the type of system. So, all participants in the creation of a business system must understand more than just the clerical aspects of their job, or the technical aspects of the job. A significant broadening of their education and thinking is essential.

4 The Four Dimensions of Software Projects

In the book *Rapid Development*, by Steve McConnell [6], it says that a system development activity includes four things that are important

- People - Tools - Process - Product

A development activity is done by PEOPLE (for PEOPLE), using development TOOLS, while following a PROCESS, to produce a PRODUCT.

Clearly, all four must be included and considered in any discussion of the successful project. However, McConnell states that

… we now know with certainty that peopleware issues have more impact on software productivity and software quality than any other factor (at p.12)

… it is now crystal clear that any organization that's serious about improving productivity should look first to the peopleware issues of motivation, teamwork, staff selection and training. (at p.13)

Many other authors have explored the importance of 'the people' in organisations and organizational activities. Senge [7], quoting Kazuo Inamori, founder and president of Kyocera, a world's leading company in advanced ceramics technology. *'whether it is research and development, company management, or any other aspect of the business, the active force is 'people'.*

Champy, in discussing reengineering management[8] states '*… responsibility and authority are so widely distributed throughout the organization that virtually everyone becomes a manager, if only of his or her own work'* (at p.70), and *'…our core values, our company culture based on trust and respect for individuals. It's about empowering people at the lowest level of the organization to run with their ideas. That freedom fosters a lot of creativity and enthusiasm'*. (at p.70), *'We just started giving out assignments and trusting people to carry them through.'* (at p.70) and *'We are all in this to build a sustained competitive advantage. You are not here solely to perform your function. You are here to add value.'* (at p.73). Finally, Champy discussed *' … 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.'* (at p.115). de Geus [9] states *'… organizations' true nature is that of a community of humans.' Arie de Geus* was the head of Shell Oil Company's Strategic Planning Group. Other articles written and published by him include 'The Living Company', published by Harvard Business Review [10] in which he uses the analogy of 'the happy family'. Of course, a family comprises people, making his implication obvious.

It is possible to find many more references in the literature to aspects of human behaviour in organizations, references to the ability of people to work independently without close supervision, to be creative, to be self-managed, and the central role that 'people' play in the success or failure of organizational processes. Developing software is a human and business activity. It is also a technical activity, but first, it is a 'human-centric' activity. Therefore, any development process will only be effective if it enables the people to perform at their best, and most creative. The development process must enable learning, and enhance the capabilities of the developers. Project management must be people-management, first and foremost, and acknowledge that the people are well-trained, competent professionals, or are at least eager and willing to learn and become so. Developers will contribute the technical knowledge to the development project, and the client, or client representative, must contribute the business knowledge. This means that the developers, and the clients, must work closely together, collaboratively, and must recognise each others' abilities and knowledge.

5 Project Management – Command and Control

The traditional project management approach is usually manifested as a 'command and control' approach to directing the lives and work of subordinates. There is a supposition, not always correct, that because the manager is the manager, and has subordinates, then the manager knows better than anyone what should be done, and, more particularly, how it should be done and how long it should take. This has too often proven incorrect and even damaging to the prospects of the business unit, and the business as a whole. In their book *The Wisdom of Teams: creating the high-performance organization*, Katzenbach & Smith [11] discuss the often damaging or even disastrous effect of an embedded 'command and control' style of management. Modern 'management' demands leadership, not control. Leadership implies providing vision, inspiring rather than controlling, showing the way forward rather than dictating the way forward. Leadership is demanded in human activities, rather than command and control style management. This is discussed further and at length in this paper.

The inspiration for this project approach has come from long-standing (and dubiously successful) project management of civil engineering and construction projects. That development model demands up-front plans, up-front requirements determination, adherence to the plan, and does not allow for adaptation of plans or the evolution of requirements. Nor does it have regular and frequent feedback cycles, relying on the project manager's estimates and plans to be correct *ab initio*. The project manager must maintain strict control over all activities to ensure their estimates are met. After all, it is the project manager who is responsible for the estimating, the planning, and the timely completion of the project, so it is the project manager's prerogative to command the troops.

6 Chaordic Systems & Ecosystems

In the landmark book Birth of the Chaordic Age [12], Dee Hock coined the term 'chaordic' to describe '*the behaviour 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 digital ecosystem is defined as an open, loosely coupled, demand-driven, domain clustered, agent-based self organised collaborative environment ... for a specific purpose or goal, and everyone is proactive and responsive for their own benefit or profit. 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 ... for a specific purpose or goal* (the development of a system)*', ' creating value by making connections through collective intelligence* (see The Wisdom of Teams)*'*.

Can these be at least some of the characteristics of the software development activity? Is it, or can it be, chaordic, inasmuch as it can be a self-governing organism, or an ecosystem that promotes collaboration and achieve a specified purpose or goal. It is suggested that this is indeed the case.

Therefore, as such, it defies a management 'command and control' style. As Champy (op.cit.[8] 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.

7 Project Success Criteria

In the traditional Waterfall Approach, every attempt is made, at the start, to impose order 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)'* (Standish Group, [13]). 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 Garmus & Herron, [14]). The study also showed that 45% of delivered projects were unusable, a dismal comment on the state of software development projects. It is possible to find a substantial catalogue of system failures that were subject to cost blowouts of the order of 10 times initial estimates and 2 or 3 times time estimates, with huge and damaging outcomes when the system failed to perform properly when 'switched on', or even total write-offs of hundreds of millions of dollars..

An important question is - Why is this? Perhaps it is because '(project management activities) *are undertaken in the mistaken belief that what sometimes works in independent manufacturing 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 - cf: 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 *'...software doesn't build, it grows and evolves'* (Arthur, [15]).

There is an interesting implication to be drawn from these statements, which can be stated in these terms - project success is measured against three standard criteria (within cost budget, within time schedule, within defined scope), yet two of those criteria are not met in at least half of projects, or, projects are evaluated in terms of how well they meet estimated cost budgets and schedule estimates, yet at least half of these projects fail to meet these estimates by a substantial margin. As for the third criteria; within scope, the Standish Group (op.cit [13]) reports that only 7% of systems delivered 100% of required features (46% of systems delivered more than 75% of required features, and overall, on average, systems delivered only 61% of required features). An interesting further consideration is that research has shown that a substantial proportion, up to 60%, of features delivered in many systems are never or rarely used by the users of the system.

8 The Failure Factors of SDLC aka The Waterfall Approach.

Many developers and project leaders now question the Waterfall Model, and it is often suggested that it is a poor model of the software development process, and too often results in project failure. Why is this?

1. **Complete Requirements Up-Front:** It depends on a full, comprehensive, complete and detailed statement of requirements, from the Client, at the start of the project. This is considered necessary so that both Client and Developer know exactly what is to be done. They are trying to achieve certainty.

   The problem is that the Client cannot specify all requirements at the start. This fact has been established in almost every software project ever attempted. This is why we have elaborate Change Management Systems in place.

   This level of certainty just is not possible, so this underlying assumption fails!
2. **Unchanging Requirements:** The assumption is that the requirements specified at the start of the project will not change during the project's lifetime, and we can deliver a system that conforms to the requirements stated at the start. The problem is that requirements do change. Clients and developers both learn many new things as the project proceeds. Things Change! Also, often what was correct and necessary at the start of the project becomes unnecessary or irrelevant because of changes to the business activities. Indeed, the longer a development project goes on, the more likely it is that the end result is unacceptable, if based

only on their initial requirements, as specified.

So this underlying assumption fails!

3. **Outdated Requirements:** To repeat; requirements do change. Like many things, they become stale while they sit on the shelf waiting to be developed. Research has shown that maybe as much as 60% of original requirements will change. This can be interpreted that 60% of the original time and effort in discovering and recording those requirements was a waste of time, money and effort. What a waste!
4. **No Adapting or Experiential Learning:** By making a plan at the start, and 'sticking to the plan', the project leader is unable to take advantage of new circumstances, or is unable to adapt to changing situations as the project proceeds. The assumption is, also, that the original plan is 'correct and workable', and if it is not, then it is a failure of project management or a failure of planning.

   The problem is that circumstances do change. Clients and developers frequently wish to change the plan, to adapt the plan, often based on the project team's experience.

   So this underlying assumption fails!
5. **No Continuous Learning:** When there is the assumption that the original specification of requirements is correct, and if any change is requested it will be viewed as a negative impact on the project plan and scope, the project leader, the client, and the developers are being refused the opportunity to discover better requirements. No learning is allowed to take place and no lessons from team experience are allowed to be included in the on-going project activity.

   So this underlying assumption is damaging!
6. **Success Criteria Limited and Inappropriate:** The conventional and usual success criteria for software development projects managed under the SDLC approach are In Time, In Budget, In Scope. It is the original projections or estimates made at the start of the project that are used in these assessments of success. But, when the success of the project is measured in terms of meeting the (original) deadline, meeting the (original) budget, and delivering the (original) specified scope of requirements, then the project is locked into a set of project success criteria, projections and plans that have almost always proven to be wrong. Delivery according to (original) scope, time and budget does not include any consideration of possible error at the start, and it also doesn't include quality, and business value of the software being delivered.

   So this underlying assumption is damaging!

   What also is missing from the success criteria of such projects is Value to the Business, and Acceptance by the Ultimate End-User. These, however, do seem to be contradictory to the acknowledged success criteria.

7. **Change Management Overhead:** As the project proceeds, any change requests are managed as if they are always potentially damaging to the project, and will have a negative impact on the project. Therefore, refusing to accept changes is considered to be an acceptable, if not a desirable, result of the 'change management' process.

   The potential result of this change management process is that every change refused moves the final delivered system one step further away from the desired and valuable system that is really needed at the time of delivery. This 'requirements gap' is at a maximum at the time of delivery and implementation, and may be so wide that the system is effectively useless. At best it may be acceptable only if there is an intention to start another 'project' for the purpose of applying all the changes to the delivered system. But this does not seem to be very reasonable or practical.

So this underlying assumption is damaging!

Conclusion: Every foundation assumption of the Waterfall Project Management Model is wrong and / or damaging to the prospect of a successful development activity. The probability of the delivered system, even one that is developed in strict accordance with the initial requirements as stated by the client, being unacceptable to the contemporary users, is very high, and is likely to be rejected.

9 The Model of Concurrent Perception

Can there be an harmonious blend of order and chaos? Rubinstein et al [16] 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' 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. Or the delivery of a system that is less than useful, and has low business value. These statistics clearly indicate a descent into chaos from a starting point of imposed order. 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 many examples of highly successful projects that seem to be well described by the Model of Concurrent Perception 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 design project 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'*. 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.

In the design project of the Lexus motor vehicle, as described in Liker [16] 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 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'*.

A much- performed activity in many projects, including software development projects, is that of brain storming. Brain storming sessions are governed by some quite strict rules, including 'At the start, every idea is accepted without argument or disagreement'. Successful brain storming meetings create the initial chaos of many ideas and suggestions, and then allow the order to emerge as decisions agreed upon. SWAT analysis can be seen also as a chaos-creating activity from which order emerges. Estimating methods such as the Wide Band Delphi [18] have an emphasis

on ‘brain storming’, or shared decision making.

10 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 (op.cit.[7]). 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, [19] 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 … favours 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 ([16], op.cit.) there are many similarities in their discussion. In 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 ([19], 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.

Another viewpoint here is what has been called ‘the wisdom of the crowd’, or ‘the wisdom of crowds’. In *The Wisdom of Crowds: Why the Many Are Smarter Than*

the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations, James Surowiecki [20] proposes the idea that the aggregation of information in groups results in decisions that are often better than could have been made by any single member of the group. Its central thesis, that a diverse collection of independently-deciding individuals is likely to make certain types of decisions and predictions better than individuals or even experts. This concept of group decision making has also been termed, somewhat inelegantly, as 'the wisdom of the herd' [21].

11 Leadership and Teams

Katzenbach et al ([11], op.cit) describe the work of a team involved in planning and implementing a major new business strategy for a large railroad company in the US, in the early 1980's. The efforts of the team were ultimately highly successful, in the face of entrenched opposition and even sabotage within the higher management of the company. The role of the leader of the team was a major factor in this success. The success of the team's project was attributed to (1) dedication to a common purpose (2) acceptance of a performance challenge (3) a sense of mutual accountability (4) candour and mutual respect between team members (5) a shared affection for each other, arising from the shared experience, and the shared success.

It is doubtful that a traditional project manager dictating the activities of the 'subordinates' in a 'command and control' style of management, could have achieved the same highly successful result. This has been the experience in many development projects, with the project manager ultimately disgraced for his 'failure' to estimate 'correctly', control rigorously, decide correctly, and designate appropriately.

Another telling case study on the idea of leadership as a success factor, published in Maani & Cavana ([19], op.cit.) is where, in 1995, a team from New Zealand won the famous and prestigious yacht trophy, called the Americas Cup. 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 team's performance surpassed any previous campaign.

How did they do it? The success has been attributed to:

- The inspirational leadership of the syndicate Leader
- The strong sense of community within the 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.

So what is a team? Katzenbach et al ([11], op.cit) again provides a suggestion. '*A team is a small number of people with complementary skills, who are committed to a common purpose, performance goals, and approach for which they hold themselves mutually responsible.*' It is suggested further that the 'small number' be optimally 10 or less.

A recent article [22] about Pixar (www.pixar.com), the organisation that develops full-length animated movies, is revealing as to Pixar's successful approach to 'product development', and its project management in a creative environment, which, as has been implied often in this article, are characteristics of software development projects. In summary, the product development approach has the following characteristics:

- a cohesive team moving from project to project. *'A team of moviemakers who know and trust one another in ways unimaginable on most sets.'*
- daily meetings where *'they ruthlessly "shred" each frame'*.
- an environment where mistakes are seen as opportunities for learning and improvement. *'We know screw-ups are an essential part of making something good.'*
- early identification of mistakes. *'our goal is to screw up as fast as possible'*.
- Upper management support and involvement. '*The upper echelons also subject themselves to megadoses of healthy criticism'*. This is leadership, not 'management'.

Clearly, this approach, which has demonstrably been shown to produce highly successful products, is everything that 'agile thinking' tries to be; highly iterative, fast feedback cycles, total transparency of progress and outcomes, self-managed, validation and verification frequently and systemically.

12 Kanban – Applying Just-in-Time Principles

Drawing from a leading book on the topic of Kanban [23] 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 favour 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 or requested 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.

13 Applicability & Relevance to Agile Methodologies

Included under the heading of Agile Methodologies for the purpose of this paper are development approaches that have been called Software Prototyping, Rapid Application Development, Iterative Development, and specifically the 'agile' approaches:

- **People Focused**: Collaborative, Self-Organizing and Self-Managing Teams.
- Empirical and Adaptive: 'empirical', 'adaptive', 'evolutionary', 'experiential' development, planning and estimating.
- **Iterative**: a series of short iterations each of which produces a useable enhancement to the system.
- **Incremental**: delivering increments to the system.
- **Evolutionary**: requirements *in detail* are continuously discovered, and are continually evolving.
- **Emergent**: The characteristics of the system emerge as parts are added.
- **Adaptive**: adaptive planning and estimating.
- **Just-in-Time** Requirements Elicitation: Requirements are stated in detail 'just in time' to develop them.
- **Knowledge-Based**: knowledgeable, self-managing team, continual knowledge sharing and learning.
- **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.

13 Conclusion

It is suggested that computer software is now so ubiquitous in everyday private and commercial life that no discussion on business, business strategies etc. can ignore matters pertaining to software. Especially in this day and age, when creativity of thought and artistic action, and the imperative of agility in system development and system operations brought about by the influence of cloud computing, SaaS, IaaS and OaaS, HaaS and Paas, a change in how system development and project management is perceived is essential, and almost a survival strategy.

Software development is seen as a chaordic activity, defying orderliness and certainty, and therefore demanding an appropriate approach to the management of that activity. A paradigm of project management that includes elements of what has been termed The Model of Concurrent Perception, the Learning Organisation, Leadership and Team Dynamics, has been discussed. The software project management approach traditionally used, based on, and inherited from civil engineering and construction project management practices is seen to have failed, as has the management model described as 'command and control'. The 'agile' approaches proffered by a growing number of experts in software development and software project management is the anti-thesis of this command and control style, and is well supported from the literature on management styles and practices, and case studies, that never seem to make it into software project management and software engineering curriculum.

The System Development Approach, whatever it may be, must necessarily must be accompanied by an appropriate project management approach. This paper has been about the Agile Development Method conforming to the Manifesto for Agile Development, as published in 2001, together with the Twelve Principles published in support of that Manifesto. This paper extends this thinking with the creation of an Agile Software Project Management Manifesto, which lists certain preferable practices and principles. These practices and principles are elaborated in Reference Disciplines drawn from predominantly management literature. These are considered valuable contributions to Agile Development and Agile Project Management thinking.

Whilst not canvassed in the paper, a final concluding comment here is the author's long held belief that these Reference Disciplines, surely amongst other subject areas, should be included in the basic education; in college and university level undergraduate programs, in the academic fields of Computer Science, Information Systems, Information Technology and Business Computing; any studies that include software development.

References

[1] R. Winston, *Managing the Development of Large Software Systems*, Proceedings IEEE WESCON, August 1970, IEEE.

[2] T. deMarco, *Structured Analysis and System Specification*. Prentice Hall, 1979.

[3] Yourdon, Edward and Larry Constantine, *Structured Design.* Prentice Hall, 1979.

[4] G. Chris and T. Sarson, *Structured Systems Analysis: Tools and Techniques*, McDonnell Douglas Systems Integration Company, 1977.

[5] http://agilemanifesto.org/ accessed 19th February, 2014.

[6] McConnell, Steve, *Rapid Development: Taming Wild Software Schedules,* Microsoft Press, Published June 1996

[7] Senge, Peter, *The Fifth Discipline – The Art & Practice of the Learning Organization*, Currency Doubleday, 1990

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

[9] De Geus, Arie, *The Living Company*, Nicholas Brealey, London, 1997

[10] De Geus, Arie, *The Living Company*, Harvard Business Review, 1997

[11] Jon R. Katzenbach, Douglas K. Smith, *The wisdom of teams: creating the high-performance organization*, Harvard Business School, 2008

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

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

[14] 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.

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

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

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

[18] Wide-Band Delphi, http://en.wikipedia.org/wiki/Wideband_delphi, accessed May 31st, 2010.

[19] Maani, Kambiz E. & Robert Y. Cavana, *Systems Thinking, Systems Dynamics*

– *Managing Change and Complexity*, Pearson Education NZ, 2007

[20] James Surowiecki, James, *The Wisdom of Crowds: Why the Many Are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations*, Random House, 2005

[21] *'The wisdom of the herd'*, http://en.wikipedia.org/wiki/Collective_wisdom, accessed May 31st, 2010

[22] Wired Magazine, *Animating a Blockbuster: How Pixar Built* Toy Story, http://www.wired.com/magazine/2010/05/process_ pixar, May 24th, 2010, accessed August 30th, 2010.

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