Agile and Lean Management – HR and IS Managers Please Take Notice[1]

Roy Morien

Faculty of Science, Naresuan University
roym@nu.ac.th

Abstract:

For more than 30 years, software development projects have been managed according to essentially civil engineering project management concepts. This has proven to be less than successful, with frequent cost overruns, missed deadlines and a large proportion of systems were either never delivered, or were un-useable when finally delivered.
During that same time period of 30 years or so, development methods were researched and published that premised delivery of quality systems, according to real requirements, at the lowest cost, and in the shortest possible time. A new paradigm of systems development method, claiming significant success for system development projects was developed published aver a period of 30 years, and received scant attention from developers and clients alike.

Agile Software Development, and the closely associated concepts and practices of Lean Product Development, encompass many aspects that are rarely seen to be appropriate or associated with software development projects. Notwithstanding the considerable success achieved by the use of these approaches, there have been many problems of implementing 'agile and lean' methods which are a result of external factors; usually or often to do with management practices and HR practices. There are often significant tensions between accepted HR practices and general management principles and practices, and agile and lean management practices. Many common threads of theory, principle and practice run through these various 'alternative' development methods, especially in the aspects of people management, organizational behavior, the 'learning' organization. Theorists and practitioners of agile and lean management also consider elements of HR practice such as motivational programs, remuneration and reward programs and performance

This paper will discuss some of the project management principles encompassed in agile and lean management thinking, and also expound on the concepts of 'the learning organization', self-directed teams, leadership styles, value stream analysis, organizational and management culture. Projects being described as 'knowledge based' rather than 'process based' will be proposed as a new view of projects which are predominantly human-centric rather than techno-centric.

Introduction

The software development project management approach, generally known as The Waterfall Approach was published circa 1977 - 1979. Structured Design Lifecycle methods were published by De Marco [1], Gane & Sarson [2] and Constantine [3]. Associated concepts of structured systems methods were published by Warnier [4], Ingevaldsson [5] about the Jackson Structured Programming method, and others. The emphasis was on 'structured', in both programming and systems analysis methods.

There has been some confusion introduced into the use of the mnemonic SDLC, which was originally the initials of the phrase Structured Design Life Cycle, arising from the publication of Structured Systems Analysis methods, and Structured Programming. However, over time its use became more to mean *the* Systems Design Lifecycle, which is a much more all-encompassing term that somehow implies that this is *the way* to develop systems. One interesting example of this confusion can be found in the curriculum statement for a subject in a university, which will remain unnamed. The subject is titled 'IS 622 Structured Systems Design' yet the first course objective states'To gain a thorough understanding of the Systems Development Life Cycle'.

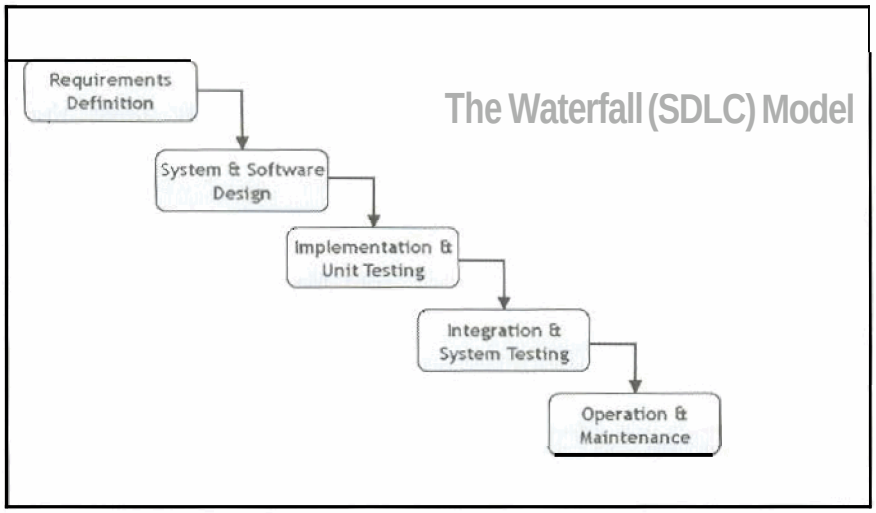
Notwithstanding the huge explosion in software development, and the ubiquitous application of computer systems software to almost every aspect of daily life, the record of software development projects remains dismal. As the IT world moved ahead at breakneck speed, many organisations and developers continued to use these development methodologies published in the late 1970s. Evidence can be found in a myriad of research papers and experience reports at conferences that clients paying for systems rarely get the value-for-money that they anticipated. Partly, this is due to the fact that there has been some considerable confusion as to what it is that a software project client is actually buying. One thing is sure; it is not a product, because at the *start* of the project there is no product, just an intention to develop one. As well, notwithstanding the many software development methods, approaches and methodologies published over the last 30 years, many of which made valiant attempts at imposing rigour and certainty on development projects, there has been a continuing record of failed projects, wasted resources and dissatisfied clients

The Waterfall Approach

The Waterfall Approach gains its name from the image of activity flowing down from one phase to the next, as illustrated in Figure 1.

[1] บทความนำเสนอในการประชุมทางวิชาการนานาชาติ ครั้งที่ 1 จัดโดย คณะวิทยาการจัดการฯ มหาวิทยาลัยนเรศวร วันที่ 5-7 พฤศจิกายน 2551

Figure 1: The Waterfall Model



Research undertaken by various organisations over the last 20 years has indicated that the greatest source of errors in delivered systems is the lack of clear and complete requirements. This seems self-evident and unsurprising, but is nonetheless of central importance. To overcome this problem, the Waterfall Approach and associated 'phased' methodologies have a Requirements Definition Phase near the start of the project. Subsequently, any changes to those requirements are seen as 'scope creep', failure to do it properly, and a risk to the good order of the project plan,

There is some apparent attraction to the idea of 'getting it right up front' and thereby knowing quite clearly and accurately what it is that is to be developed The theory is that what happens after this requirements analysis and discovery phase is driven by the intention to manifest those requirements m the final system, which will be carefully planned in detail, and that plan will dictate the progress of the project.

Unfortunately, this has rarely been the case. There is considerable criticism of this approach and the serial, phased approach generally, which is seen to manifest a high level of risk, and a danger to the success of project. The reasons for this include:

1. There is considerable difficulty in ascertaining all of the requirements at the start of the project Clients usually just cannot tell it all.
2. It is an impossible task to *capture* the requirements in *such* detail that there is no doubt about exactly it is that must be developed.
3. By apparently capturing all requirements at the start, and 'freezing the specification', no further opportunities to learn about real needs and requirements are available. The fact that project participants can and do learn significantly more about requirements, about possible features, as the project proceeds, is ignored.
4. Having lengthy phases with 'sign off' at the end of each phase, gives little opportunity for feedback and the verification and validation of the development progress, and the correctness of what is being produced. *This* flies in the face of accepted systems theory relating to system feedback loops.
5. Having the major QA activity at the *end of* the process is *seen* as allowing the accumulation of errors and the further promulgation of those errors through the system, and a significant amount of rework at the *end* of the project.

One outcome of this approach is that, instead of the project manager becoming more confident about the success of the project and about the implementation, they become increasingly less confident as the number of changes requested and rejected grows, and the size and complexity of the system grows without timely and frequent reassuring feedback and validation. It seems that, often, the greatest time of anxiety and tension for

software project managers is immediately before the system 'goes live'.

Basically, it can be stated that the pre-planned, rigorously pre-defined project is at the greatest level of risk of failure, from the start, and bears the seeds of its own destruction.

Agile & Lean Development

Software development has been described as a 'chaordic' activity. Chaordic refers to a system that blends characteristics of chaos and order. The term was coined by Dee Hock [6]. Chaordic has also been defined as an adjective referring to the behavior of any self-governing organism, organization or system which blends elements of order and chaos. Chaordic organization is one able to maintain a harmonious order-disorder balance, characterized by principles of evolution; its nature includes being self-organizing, self-governing, adaptive, and nonlinear. Software development projects are seen to be chaordic activities, balancing chaos and order with a high tolerance for errors as an opportunity to learn and develop. This is actually a quote drawn from Rubinstein & Firstenberg [7] who apply it organizations 'alive with the ideas and commitment of its people'.

Such an activity as software development therefore cannot be controlled and managed as being process-oriented. It defies the imposition of 'process'. The traditional software project management methods have tried to achieve this, but have substantially failed, All attempts at eradicating uncertainty and 'chaos' by the imposition of overwhelming 'order' Rave been unsuccessful, in the main and are fundamentally misguided and impractical. One fundamental 'Law of Nature' is that the future is, and will always be, uncertain. Events even tomorrow cannot be planned with 100% certainty, and events that are planned to happen in a certain way, at a certain time, with a certain outcome, a year or two years from now are so uncertain as to be nonsensical to try.

A 'new' approach to software systems development has been suggested, known as 'Agile Software Development'. Agile software development methods have actually been around for 20 years or mare, albeit under different guises, but have received scant attention from an industry that has been set on commanding and controlling the software product development activity, notwithstanding that this has clearly flown in the face of process theory. A 'chaordic' activity defies 'process' and 'command and control' management. It is not a process. Software development can better be described as a learning-based, or knowledge-based, activity.

The primary 'resource' in software systems development is 'the people'. It is all about 'the people', McConnell [8] suggests 4 dimensions of system development: People, Tools, Process, and Product. A development activity is done by PEOPLE (for PEOPLE), using development TOOLS, while following a PROCESS, to produce a PRODUCT. McConnell states (at page 12) *'we now know with certainty that peopleware issues have more impact on software productivity and software quality than any other factor.* And at page 13 *'it is now crystal clear that any organization that's serious about improving productivity should look first to the peopleware issues of motivation, teamwork, and staff selection and training.* Other significant authors in the information systems field have discussed the significance and relevance of 'people' in the development process (see Demarco & Lister [9], Constantine [10], Plauger [I 1]. Looking outside the information systems field to the business management literature, we can see suggestions and assertions about 'people' and 'people management' that are extremely apt to the software development situation. Champy [12] states *'... You cannot have a culture of obedience to chains of command and the job slot. It just won't work'* and *'... 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.'* We can also see *'Maintaining a qualified IS workforce has been identified as one of the top ten concerns of IS executives recently (and similar human resources issues were among top concerns in prior* key issue *studies over the last two decades'[13].*

This is not, however, a recent surge of opinion about the 'people' aspect sf systems development, As far back as 1988 it was stated that the IS practitioner needs skills in the following areas: Technical Skills, Human Resource Skills, Business Knowledge, and Transitional Skills. It is this latter skill that holds particular interest. To quote the publication, *'Transitional skills are deemed necessary because* system *professionals face drastic changes, brought about by business changes and more flexible and* easy-to-use *technology'* and *'Transitional training will emphasize understanding and effectively dealing with change, migrating to new technologies, learning to work closely with people in other areas of the company, understanding organizational behavior, and broadening technical knowledge' [14].* Urquhart [15] concluded that *'Given the documented failure of projects and evidence of communication problems in the software definition*

stage, ... strengthening developers' personal skills would make at least as a valuable contribution to project success as the adoption of system development methodologies'. (Originally cited in Morien & Schmidenberg [16]).

Since the early 1980's there have been many research papers and articles published at conferences and in journals about software development methods that have been seen as more appropriate to a 'people oriented, chaordic' activity; specifically in this context software development. Methods such as Software Prototyping (Nauman & Jenkins [17], Bemelmans[18], Budde & Kuhlenkamp [19]), Evolutionary System (Hawgood [20], Rapid Development (Martin [21]), and of course Rapid Applications Development (McConnell op.cit [8]). It can be said that many of these development approaches have converged into what is now called Agile System Development (Agile Alliance [22]) and this 'movement' has spawned a number of methods and approaches that are basically iterative, adaptive and empirical in nature, much better suited to software development, if seen as a chaordic, learning and knowledge-based activity, undertaken by people. Popular agile methods includeEVO (www.xs4all.nl/~nrm/EvoPrinc/), Extreme Programming (www.xprogramming.com/), Scru(www.controlchaos.com), Crystal(alistair.cockburn.us/index.php/Crystal_methodologies_main_foyer), Feature Driven Development, (www.featuredrivendevelopment.com/), Dynamic Systems Development Method (DSDM) (www.dsdm.org/), Lean Software Development (www.poppendieck.com/) Agile Unified Process (www.ambysoft.com/unifiedprocess/agileUP.html).

Agile Development

The Agile Manifesto [23] states the essential guiding principles of agile methods. These are elaborated upon, and their implications are focused by a number of authors. Evans [24] suggests that the essential difference between the traditional approaches to systems development and the agile approaches is the difference between planned iteration and the unplanned rework so common in waterfall-based projects. With the traditional approach, adherence to the prescribed software process is considered the major determinant of success. With the agile approach, adaptation toward achieving the end-goal – working software – is the major factor in success. This table, taken from Evans (op.cit) summarizes some *'contrasts between these remarkably different approaches.'*

	Waterfall	Agile
Guiding metaphor	Manufacturing / Engineering	Organic /Emergent
Focus	Documentation, Schedule	People, Working Code
Dynamic structure	Cause and Effect, Preventive Approach	Chaordic (Ordered Chaos), Adaptive Approach

Welcome Change

If there is one guiding principle of agile development, it is 'Welcome Change'. This implies the acceptance of the fact that requirements in detail cannot properly, comprehensively or accurately be defined at the beginning of the project (the 'Big Bang' approach, or the Big Design Up Front (BDUF) approach [25,26]), and are almost certainly subject to change in extended period projects. Highsmith [27], in Orr [28]) states that *'By the time a three-year project delivers its first working versions, many of the users have forgotten what they agreed on in year one or have moved on so that the people who have to work with the system have little or no idea what it was developed for'* Ha further suggests that *'if a project takes three years to implement, you can be sure that the requirements will be at least two years out of date by the time it comes into existence.*

Specific definitions of agile development have been attempted by Mahanti [29] as *'a departure from plan-driven traditional approaches, where the focus is on generating early releases of working software using collaborative techniques, code refactoring, and on-site customer involvement'*. And Melnik and Maurer [30] as *'human centric bodies of practices and guidelines for building usable software in unpredictable, highly-volatile environments'*. Software development projects are considered to be an unpredictable and highly volatile environment.

For the purpose of this discussion, a definition of Agile and Lean development is '*A software development method is said to be an agile software development method when a method is people focused, communications-oriented, flexible (ready to adapt to expected or unexpected change at any time), speedy (encourages rapid and iterative development of the product in small releases), lean (focuses on shortening timeframe and cost and on improved quality), responsive (reacts appropriately to expected a d unexpected changes), and learning (focuses on improvement during and after product development)*' [31]

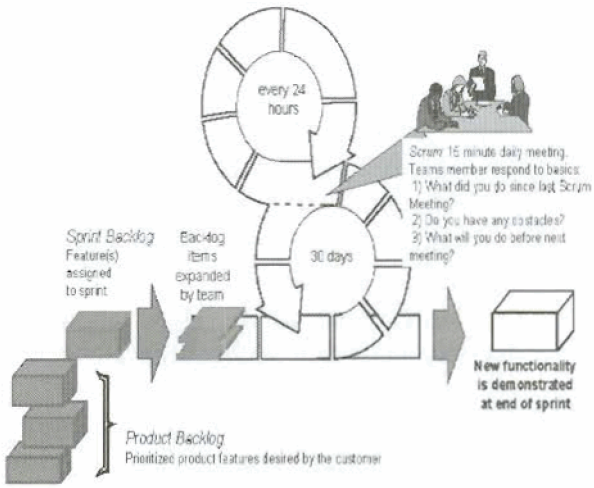
So, for a software development activity to be *agile,* it should encompass practices that can be variously described as:

- 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 the Project Manager, to decide on the work to he 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-
- Iterative: Development is achieved through a series of short iterations each of which produces a useable enhancement to the system.
- 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 m 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.

Scrum agile method emphasises project transparency, continual communication and collaboration between project partners. The Scrum method is illustrated in Figure 2 (from www.controlchaos.com)

The Scrum method depends on a Product Backlog that is a prioritized list of all the requirements as they are known and understood at any given time. There is no expectation of a complete and comprehensive list. There is an expectation of change, however. The Scrum project starts with analyse meetings between all players to elaborate the Product Backlog as much as is possible at that early time in the project. The project proceeds in a series of short iterations, called Sprints in this method.

Figure 2: Scrum Method Diagram



Sprints can be as short as a week, and as long as a month. Shorter sprints are suggested to keep the project highly visible to all players, and then maintain a rapid and constant output of useable components. (The diagram suggests a 30 *day* sprint, but there is a preference for a weekly sprint). At the start of each sprint a full team meeting takes place, and team members volunteer for tasks as stated on the Project Backlog - the highest priority tasks *being* taken first.

To maintain project impetus, and high transparency and visibility, there is a formal Daily Scrum, or Daily Standup Meeting, where all members of the development team share their dy's experience. This is not a 'report to the Project Manager' meeting, but a collaborative, knowledge

sharing, help-seeking (where necessary) meeting. It is intended to not last more than 15-20 minutes, but is essential to keep all developers 'in the loop' and fully aware of the project activity.

At the and of each sprint, the completed outcomes of the sprint are demonstrated to the client, and if possible released into the production environment.

The Benefits of Iterative Development

Agile methods are essentially iterative and incremental. Recommended practice is that iterations be no longer than 4 weeks, with many practitioners *tending* to agree on 1 week iterations. Iterative development has a number of advantages:

- Frequent opportunities to validate and verify requirements,
- Frequent opportunities to learn and share knowledge. Iterative development assists in the education and learning process. In the traditional approach there is little opportunity to learn abut real requirements, beyond the fist major Analysis phase.
- There is always a clear deadline a short time in the future. A constant and consistent level of effort is encouraged in this way.
- The progress of the project is always highly visible to all concerned
- Confidence built in the client about the development teams ability to perform and deliver
- Detailed requirements are always 'fresh' (Just-in-Time Analysis).
- Agile practice requires continued focus on selected components during a Sprint ... no task swapping and disruption of team activity from 'special requests' or demands during the Sprint
- Team 'velocity' or rythm (also called 'takt time') can be established, and final deadlines continually known. Agile approaches can be sometimes described as deadline driven, or budget driven
- This means that when the time or cost budget is reached, all that mains are low value requirements.
- Maximum wastage on failed development is maybe 2 weeks, or 30 days maximum.
- Risks can be identified early, and immediate action taken. Fail early, rather than later.
- In an agile, iterative development activity, requirements of greatest business value are delivered first.

Lean Software Development

Alongside 'Agile Development' there has evolved another view of systems development known now as Lean Software Development.

Lean Software Development has grown out of the Lean Product Development thinking made famous by the Toyota Company. This approach became known as The Toyota Way, and had assisted in the Toyota Motor Company becoming the most efficient, most profitable and most quality conscious manufacturer of motor vehicles in the world. Of primary importance to us in our thinking about software development is the concept of 'waste' in our activities Lean 'thinkers' are continually and aggressively identifying and eradicating *waste* in processes.

Value Stream Analysis

A central theme in Lean thinking is the idea and activity of Value Stream Analysis. In simplest terms, this is the inspection of every step in a process, identifying exactly what adds value to the end product, and what does not. What does not add value to the end product may be euphemistically termed 'administrative overhead', but in reality it is 'waste'. Presumably some 'waste' cannot be avoided, but in the main, it is 'waste' that makes a process inefficient and at least partially ineffective. It is suggested that any 'software process improvement' initiative that does not subject any process to Value Stream Analysis will fundamentally fail to produce *any* 'improvement'.

An inspection of a development process as used by a development team may reveal many aspects of 'waste'. For example, when one developer is responsible for analyzing and defining an Entity, and then passes all the documentation to another developer to design the database table, who *may* then in turn pass some documentation to a third developer to create the interactive processing screens, there is a lot of wasted knowledge here, and a lot of wasted time, This is because the 2nd developer must learn what the first developer already knows (thus duplicating the effort) and may in fact not learn as much as the first developer (thus losing knowledge, which was hard-won by the first developer. Every time there is a hand-off from one person to another there is inefficiency and 'waste' implied *in* that action.

When we spend a lot of time analyzing and recording requirements at the *start* of a project (as is required in The Waterfall Approach to development) the reality is that something over 50% of those requirements will change. We have therefore spent 50% of that time *on* wasted effort, and must discover the 'same' requirements again (actually discovering the different requirements). If we persist in developing according to the original specification, we are in reality wasting valuable development time and effort, because either the User will reject those features of the new system, and / or we will have to throw that away and write some different software. Either way it is a waste of time.

Having requirements sitting *on* the shelf far months *or* even years waiting to be developed is a waste. Having software sitting around waiting for someone to check it and test *it,* is a waste. Writing software that users really don't want or need is a waste. Handing off partially completed specifications and code to another developer, *or* tester, is a waste. Waiting for the user to review the delivered software and give us feedback is a waste. Waiting for the QA department to do testing *on* our software is a waste.

Another important source of 'waste' in the Waterfall, phased approach is the handing off of the outputs of each phase to the specialist actors in the next phase. The Business Analysts and Systems Analysts responsible for the production of a usually large and complex Requirements document hand it off to the specialist designers, who must spend significant time learning what the previous actors already know. Then the designers, having added their extra requirements, hand it off to the developers, who must in their turn learn what the previous actors already know.

There are so many potential 'wastes' that *can* be identified in a systems development project. *A* waste is defined as some time expended, *or* some activity undertaken, or some code produced, that does *not* add value to the product. Administrative overheads are often identified as waste, for example, because they add nothing to the finished product.

Lean software development has been written about in two books specific to the topic (Poppendieck & Poppendieck [32]. [33]]. In those books, a set of Software Development Wastes was described, derived from the 7 Wastes of Manufacturing which were originally specified by Shigeo Shingo [34], one of the masterminds of the Toyota Production System. The Toyota Production System is described and discussed at length as The Toyota Way in a book on the subject [35]. The transition of the 7 Wastes of Manufacturing into the 7 *Wastes* of Software Development was made in the first book *on* Lean Software Development, and revised somewhat in the second book. For completeness I include both tables here.

It is not my intention to include a treatise of Lean Product Development or Lean Software Development here. There are many books published and many organizations in existence what are informative about 'lean'. But *to* make the point about Lean Software Development, i will discuss some of the aspects *of* Waste directly from the Poppendieck's books, as

The 7 Wastes of Manufacturing	The 7 Wastes of Software Development (from the first book)	The 7 Wastes of Software Development (from the second book)
Inventory	Partially Dona Work	Partially Done Work
Extra Processing	Extra Processes	Relearning
Overproduction	Extra Features	Extra Features
Transportation	Task Switching	Hand Offs
Waiting	Waiting	Delays
Motion	Motion	Task Switching
Defects	Defects	Defects

The Model of Concurrent Perception

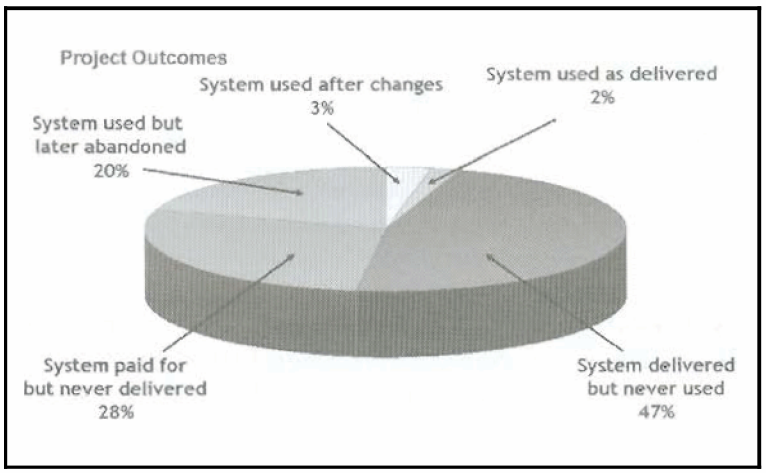
In their book *The Minding Organisation*, Rubinstein and Firstenberg describe what they term *The Model of Concurrent Perception.* This describes most eloquently the characteristics of participant behavior in chaordic systems, I would

suggest (although these authors do not themselves make this association). Their discussion suggests that 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'*. They suggest that *'chaos should be deliberately created up front'*. By this they basically 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 he 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. This 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'*. 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. Figure 4 shows the distribution of software system project outcomes. (Unfortunately the source for this data has been lost).

This data is acknowledged as having come from one particular source, but other research projects have arrived at similar conclusions, but with different percentages.

These outcomes, which show that only 2% of systems were used as delivered (and presumably as originally specified, but this is not stated), and 28% of systems that were paid for but were never delivered, and 47% of delivered systems were never used, clearly indicates a descent into chaos. 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'*.

Figure 4: Distribution of Software System Project Outcomes



Examples of highly successful projects that seem to be well described by *the* Model of Concurrent Perception include the development by the Boeing Corporation of the 777 airliner, and the development of the Lexus luxury motor vehicle by the Toyota Company.

Described by Poppendieck [33], the development of the 777 airliner had the following project characteristics:

- A totally new aircraft design
- High level of collaboration with the customer
- Tight development time-line
- Root cause of delays and problems in previous design activities identified and analyzed:
  - People not working together,
  - No culture of looking far problems – leave it QA at the end of the line,
  - Lack of prompt and effective communication between developers

o Essentially People Problems

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 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 [35] 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'*. It is a well known fact that the Lexus motor vehicle quickly became a very popular mode in the marketplace. Analysis of this situation allows us to suggest that the 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'*; or, to be a little whimsical here, they ended up with *many* orders, making the Lexus a highly successful product.

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 [36]. 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, [37] 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 ([7], 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 lap 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 [37], 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.

It's the People

So what does this tell us about software projects? In fact, a new paradigm for software system development projects is being elucidated here; that is, the chaordic, Model of Concurrent Perception, where 'people' *are* the *central* focus, the central players,. So, first and foremost, it *is* the people involved, and the way in which they can and will behave, interact and cooperate that demands attention. Software developers work in that environment that has been described here as chaordic, or, stated another way, in which the system operates on the edge of chaos, continually converging on order The Learning Organization is

an apt description or exemplar of that activity. Developing software is essentially a creative activity. However, to maintain creativity and continual improvement, the *system* continually attempts to diverge back *to* chaos, and again subsequently converges *on* order, as a continuing cycle.

Reference has been made, previously, to the importance of the people in the activity (see Champy [12]). This view is reinforced by, for example, Senge [36], 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". ...*

For the HR department, it is incumbent on them to understand and support this cycle of creativity, chaos and convergence on order by hiring the right people. It can no longer be tolerated to hire highly technical people who are incapable of the "Human Resource Skills, Business Knowledge. Transitional Skills' that were referred to so long ago in Urquhart [15].

For the IS function, for its management the software development activities, they must understand that 'command and control' management style is fundamentally irrelevant, unsuccessful and even damaging to the software development activity, given its essential 'chaordic' nature. For the developers themselves, they must be able and willing to demonstrate the essential characteristics needed in such a 'chaotic' activity, such as ability to cooperate, ability to collaborate, willingness to self-manage, self-discipline and a high level *of* goal-seeking, to achieve the convergence onto order that is inherent in the success of such projects.

There are Lessons to be learned by universities and colleges here as well. To put it simplistically, I suggest that Computer Science courses cease teaching compiler design and allow their students to *study* The Toyota *Way of* product development. It is certainly about time courses in software project management discarded their roots in civil engineering project management, and adopted the theories and principles of lean product development, and the practices of 'agile project management'. However, without an understanding of the true essential nature of such activities, this is unlikely to occur,

An anecdote is germane here, I believe. During a study tour of universities in Thailand (reported in Morien [38], [39]) a young Thai academic demonstrated to me a website generator that he had developed. As I had had a long-standing interest in such development tools, 1 viewed this software product with considerable interest; and was highly impressed with it. I asked this young academic the following questions, and received the following answers; *Q:* How long did this take you to develop? (A; 5 years). Q: Did you know exactly, at *the* start, what you would develop, and write a detailed statement of requirements and features at that time? (A: No, of course not, I only had a good idea, but I didn't know then exactly what I would include). *Q:* Did you learn more and more about what you wanted to include in the product, as you proceeded with the development? (A: *Yes*, every time I did my development, I learned more about what I wanted). *Q:* Did you create a detailed plan, at the start, using perhaps Microsoft Project to create a Gantt Chart? (A: No, that would have been impossible and useless anyway). My final rejoinder to him was 'Well why do you teach your students the Waterfall, phased approach to software development, when you have just demonstrated clearly to yourself that it is not practical or useful? (A: slightly embarrassed silence),

Conclusion

Software development is an activity' that defies 'command and control' management styles, and rigorously pre-planned project activity. Reference to other disciplines in the area of product development, lean thinking, and organizational and human behavior in groups provides us with a substantially better model of systems development that the traditional IS project management literature.

The best of these models is, it is suggested, the Model of Concurrent Perception, which can be applied to many famous product development projects, at least in retrospect. The software development modal that best fallows these principles is what are now termed 'agile development' approaches, which are seen as the successors of software development methods that have been researched and published *since* the early 80's, and which have substantially been subsumed into the agile development model.

The substantial history of 'failure' in software projects, with a price tag of literally billions of dollars, makes it incumbent upon various function areas of organizations to acknowledge this, and to seek better ways of development, and more suitable personnel to do it. HR and IS need to *see* this *common* vision, and educators need to change the content of their curriculum to allow students to

discover and learn about the 'alternatives', which are fast becoming mainstream, leading educators by a long way.

References

[1] De Marco, Tom. (1979). **Structured Analysis and System Specification**, Upper Saddle River, New Jersey: Prentice Hall.

[2] Gane, Chris and Sarson, Trish. (1979). **Structured Systems Analysis: Tools and Techniques.** Prentice-Hall.

[3] Constantine, Larry L. (1479). **Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design**. Yourdon Press.

[4] Warnier, Jean Dominique (1976). **Logical construction of programs.** New York: Van Nostrand Reinhold.

[5] Ingevaldsson, Lief. (1980). **Jackson Structured Programming A Practical Method of Programme Design**. Chartwell-Bratt.

[6] Hock, Dee. (1999). **Birth of the Chaordic Age.** San Francisco: Berrett-Koehler.

[7] Rubinstein, Moshe F. and Firstenberg, Iris R. (1999). **The Minding Organisation.** New York: John Wiley & Sons.

[8] Steve McConnell. (1996). **Rapid Development: Taming Wild Software Schedules.** Microsoft Press.

[9] Demarco, Tom and Lister, Timothy. (1987). **Peopleware: Productive Projects and Teams.** New York Dorset House.

[10] Constantine, Larry L (1995). **Constantine on Peopleware**. Englewood Cliffs, New Jersey: Yourdon Press.

[11] Plauger, P.J. (1993). **Programming on Purpose 11; Essays on Software People**. Englewood Cliffs, New Jersey Prentice Hall.

[12] Champy, James. (1995). **Reengineering Management: The Mandate for new leadership.** HarperCollins.

[13] Lending, Diane and Chervany, Norman L. *The changing systems development job' a job characteristics approach*, Special Interest Group on Computer Personnel Research Annual Conference. Proceedings of the 1997 ACM SIGCPR conference on computer personnel research, San Francisco, California, United States. (pp. 127 – 137)

[14] I/S Analyzer (1988). *Preparing For Tomorrow's Systems Jobs*, Vol.26, No. 5, May

[15] Urquhart, C. (1993). *Changing Skill requirements in Information Systems - A Skills Model for the Systems Analyst*, Proceeding of the 4th ACIS, University of Queensland, September.

[16] Morien, Roy and Schmidenberg, Olive. (1994). *Educating Information Systems Professionals: The Tertiary Educational Challenge*. The Proceedings of apitite'94 (Asia Pacific Information Technology in Training and Education). Brisbane, Australia, June.

[17] Naumann, J.D. and Jenkins, A.M. (1982). Prototyping: The New Paradigm for Systems Development. **MIS *Quarterly***. Sept., Vol.6, No.3.

[18] Bemelmans, Th.M.A. (ed) (1983) **Beyond Productivity: Information Systems Development for Organisational Effectiveness.** North-Holland.

[19] Budde R., Kuhlenkamp K., et al. (1984). **Approaches to Prototyping.** Springer: Verlag.

[20] Hawgood, J. (ed). (1981). **Evolutionary Information Systems**. North-Holland.

[21] Martin, J (1991). **Rapid Application Development.** McMillan.

[22] Agile Alliance, (2008). http://www.agilealliance.org, Accessed October 1st.

[23] Agile Manifesto. http://agilemanifesto.org, Accessed October 1st.

[24] Evans, Ian. (2006). *'Agile Delivery at British Telecom, Methods & Tools'*, Summer 2006:20, www.methodsandtools.com/mt/download.php?summer06.

[25] Ambler, Scott. (2003). **Agile Database Techniques: Effective Strategies for the Agile Software Developer.** Wiley Application Development Series.

[26] Ambler, Scott (2003). *'Something's Gotta Give'*, Dr Dobbs Architecture & Design, March 1st, 2003. http://www.ddj.com/architect/184414962.

[27] Highsmith, Jim. 'bio' http://www.adaptivesd.com/about.html, accessed October 2007.

[28] Orr, Ken. (2002). 'Agile Requirements', Agile Project Management Advisory Service, Executive Report, Vol. 3, No. 12, Cutter Corporation.

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

[30] Melnik, G. and Maurer, F. (2005). *'Agile methods: A cross-program investigation of student's perceptions of agile methods'.* Proceedings of the 27th international conference on Software engineering ICSE'05, ACM Press, IEEE Computer Society.

[31] Qumer A., B. Henderson-Sellers, (2007). *'An Evaluation of the Degree of Agility in Six Agile Methods and its Applicability for Method Engineering'*, Information and Software Technology.

[32] Poppendieck, Mary and Poppendieck, Tom. (2003). Lean Software Development: An Agile Toolkit. The Agile Development Series. Addison-Wesley Professional.

[33] Poppendieck, Mary and Poppendieck, Tom. (2007). Implementing Lean Software Development: From Concept to Cash. Pearson.

[34] Shigeo Shingo. (n.d.). Study of 'Toyota Production System'. (n.p.).

[35] Liker, Jeffrey K. (2004). The Toyota Way. McGraw-Hill.

[36] Senge, Peter M. (1990). The Fifth Discipline – The Art & Practice of the Learning Organization. New York: Currency Doubleday.

[37] Maani, Kambiz E. and Cavana, Robert Y. (2007). Systems Thinking, Systems Dynamics – Managing Change and Complexity. New Zealand: Pearson.

[38] Morien, Roy and Orasa Tetiwat. (2007). Agile Software Development Methods Adoption in Thailand – A Survey of Thai Universities. The Proceedings of ISECON 2007. Pittsburgh, October 2007.