Complexity theory as a model for the delivery of high value IT solutions

Baden Wehmeyer

Thesis presented in partial fulfilment of the requirements for the degree of Master of Philosophy
(Information and Knowledge Management)

STELLENBOSCH UNIVERSITY

SUPERVISOR: Mr D F Botha

March 2007

Declaration

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Signed: ……………………………… Dated: ……………………………..

Abstract

Many variations of Systems Development Life Cycle models have evolved over the last fifty years of systems engineering and software science, yet not enough knowledge is available to better understand these as Complex Adaptive Systems by studying chaos and complexity theories. The primary application domain of the thesis is focused on the development of electronic hardware and software products.

There is a great need for innovation to reach all corners of the development ecosystem; however a large cognitive distance exists between the concept of systematic product development and that of value creation. Instruments are needed to aid process agility, for defusing imminent problems as they mount, and for making effective decisions to sustain maximum productivity. Many of these objectives are neglected in systems development practices. As with so many management fads, it appears that no single one of these models lived up to all of the expectations and in many cases ended up being recipes for disaster.

The statistics available on failed projects are concerning but has not stopped the scientific and engineering communities from trying over, and over again, to make progress. The goal of the thesis is therefore to identify the most viable model that supports the sustainability of systems development team performance. The research draws insights from extant literature, by applying a knowledge management theory based analysis on the various models with specific attention given to complexity theory.

The dominant metric discovered is to measure the Value Velocity of a Systems Development Team. This metric is determined by two independent variables, being Value Created and Delivery Delay.

Complex Adaptive Systems simply requires a guiding vision and a carefully selected set of generative rules for increasing and sustaining the Value Velocity.

Opsomming

Menige variasies van stelselsontwikkelingsmodelle het ontwikkel oor die afgelope vyftig jaar in stelselsingenieurswese en sagtewarewetenskap, en steeds is daar nie genoegsame kennis beskikbaar om beter begrip te kry oor hoe hierdie stelsels as Komplekse Aanpassende Sisteme bestudeer kan word nie, ten einde die bestuur daarvan te verbeter. Die primêre toepassingsgebied in die tesis is gespits op die ontwikkeling van rekenaarhardeware en -sagteware.

Die behoefte vir innovasie moet al die fasette van die ontwikkelingsekosisteem bereik. Die bewusheidsgaping tussen sistemiese produkontwikkeling en waardeskepping, is te wyd. Instumentasie word benodig om te help met ratsheid in prosesuitvoering, om dreigende probleme te ontlont, en effektief besluitneming toe te pas, en sodoende produktiwiteit op ‘n maksimum vlak te hou. Hierdie doelwitte word tot ’n meerdere mate in die huidige praktyk verontagsaam. Net soos somige bestuursadvies oneffektief is, blyk dit dat daar nog steeds geen stelselsmodelle is wat alle verwagtinge bevredig nie. In baie gevalle eindig die toepassing daarvan in waan en mislukking.

Die statistiek beskikbaar op mislukte projekte is onrusbarend, tog het dit nie vooruitgang gekelder nie, en die behoefte na verbetering bestaan steeds. Die doelwit van die tesis is dus om die mees lewensvatbare model wat die voortbestaan van stelselsontwikkelingsgroepe sal kan handhaaf, uit te sonder. Die navorsing neem insigte uit hedendagse literatuur en is gebasseer op ’n analiese van verskeide kennisbestuursteorieë teenoor die bestaande stelselsontwikkelingsmodelle. Die fokus is meer spesifiek toegespits op kompleksiteitsteorie.

Die hoofmaatstaaf is om die Waardesnelheid van ’n stelselsontwikkelingspan te bepaal. Hierdie maatstaaf word gepyl deur twee onafhanklike veranderlikes, naamlik die Waarde Geskep en die Afleweringsvertraging.

Ten slotte, vereis Kompleks Aanpassende Sisteme slegs die aanwesigheid van 'n leidende visie tesame met 'n goeddeurdagte stel ontwikkelingsreëls, wat aanleiding sal gee tot die verhoging en behoud van die Waardesnelheid.

Contents

Figures

Chapter 1
Problem description

If a man would persist in his folly, he would become wise.
William Blake

The chapter defines the scope of the research and introduces the problems encountered by systems development teams that are responsible for the reliable delivery of final valuable IT solutions. The research objectives and methodology is described.

1.1 Introduction

The management of the development life cycle for complicated systems is a complex endeavour. Some of the attributes that makes it so complex is the *unknown requisite variety*[1] while these systems are still under development. The complexity is exacerbated by unexpected events in the operational environment, changing expectations of the various stakeholders as well as unforeseen interventions and factors such as ambiguity, excessive workload, blockages in information flows, lack of essential innovation and disruptions in the development ecosystem.[2]

Many systems development methodologies and models used are often too specific, not future-proof and not well suited to provide teams with a competitive advantage at the leading edge of progress. However, if the models are too abstract it renders itself equally useless to be insightful and practical enough for general adaptation. Metaphors, models, typologies and taxonomies, all impose certain limitations on the practice of making sense and being wholly mindful of the problem domain.[3]

[1] Ashby, W.R. 1958. Requisite Variety and Implications for Control of Complex Systems. Cybernetica, Vol.1:83-99.

[2] Heylighen, F. Joslyn, C. 2001. Principia Cybernetica Web. The law of requisite variety for Cybernetic systems states that the controller must have a sufficiently large variety of actions in order to ensure a sufficiently small variety of outcomes in the essential variables. The variety of influences a system can potentially be confronted with is unlimited, and therefore the goal would be to maximise the internal variety of the variables, so as to be optimally prepared for foreseeable or unforeseeable events.

[3] Weick, K.E. 1995. Sensemaking in Organizations. According to Weick the discipline of making sense is one of placing frames around problem situations, and seeing patterns, in the pursuit of mutual understanding.

1.2 Research objectives

There are many System Development Life Cycle (SDLC) models and methodologies. These models and methodologies are predominantly focused on process descriptions and conformance to standards. There is however a great need for innovation to reach all corners of the development ecosystem and to bridge the chasm that exists between the concept of systematic product development and that of *value creation*.

Such a complex phenomenon is possibly best modelled as a *black-box*, by applying empirical management style. The approach would however require various instruments for measuring the performance of these black-box based systems. A management dashboard fitted with several instruments is required for measuring process agility, for identifying and defusing imminent problems as they occur, and for sustain maximum productivity.

The research objective is therefore to identify the most viable model that supports the control of, and sustainability of, systems development performance. To measure performance would require a clear definition of what performance is. Performance is a subjective concept that requires a mutually acceptable definition that is measurable with achievable targets. A proposed measure for performance is Value Velocity measured by the Value Created over a Delivery Delay.

The hypothesis of the thesis is that satisfactory system development productivity is achievable by modelling the system development team, the primary unit of production, after that of Complex Adaptive Systems.

1.3 Research methodology

The reflective research methodology applied is similar to that used by Cockburn as depicted in Figure 1-1. A broad literature survey directed by a search for the required attributes of extant SDLC methodologies and management theories that support the research hypothesis.

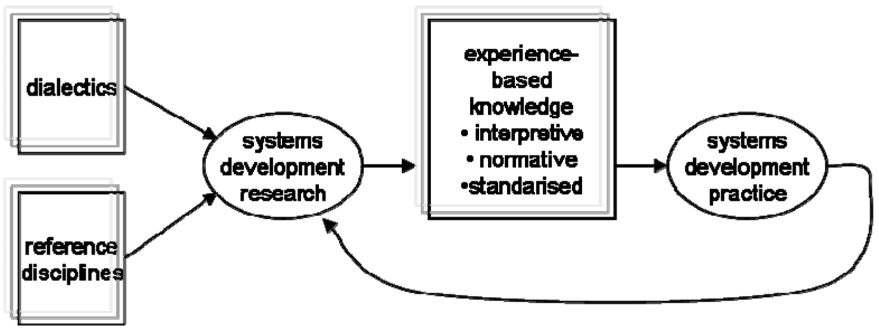

dialectics
reference disciplines
systems development research
experience-based knowledge
• interpretive
• normative
•standarised
systems development practice

Figure 1-1: Mathiassen's reflective systems development research methodology[4]

Figure 1-2 depicts the inductive approach to searching for a model that fits the problem domain. The predicated solution will search the problem domain at various a priori levels such as a) known formal models, b) intersections, and c) unexplored models.

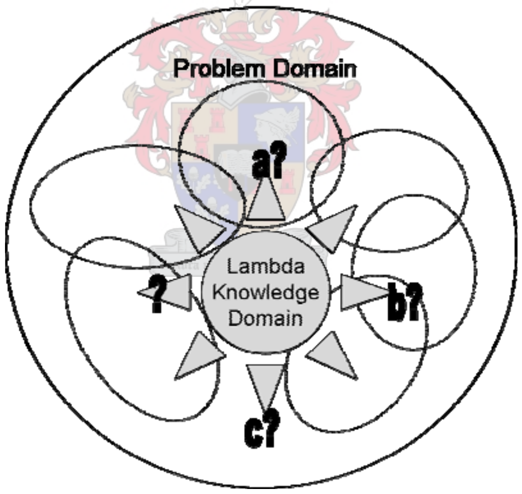


Figure 1-2: Exploring the problem domain by applying inductive analysis

Lambda is used here in the same sense as used in lambda calculus as is commonly found in mathematics and computer science language representations as a placeholder that symbolically represents an extant *and* future concept. Robust solutions should be designed with plausible future scenarios in mind. The aim is therefore to define a solution that can be represented as a universal template that is validated through *induction*.[5]

[4] Cockburn, Alastair. 2003. People and Methodologies in Software Development.

[5] Inductive analysis is a qualitative approach to problem solving that places a specimen with the frame or context of the proposed theoretical model.

1.4 Detailed problem description

In his well-known paper *No Silver Bullet*,[6] Frederick Brooks addresses the problem of missed schedules, blown budgets, and flawed products. Brooks reasons that *essential complexity* needs to be mastered to avoid conceptual errors as early as possible and that *accidental complexity* is under control. Brooks warns that the extreme costs of not affording enough time to designing architecture before construction commences.

Many past innovations in the software sciences such as structured programming languages, object-oriented programming, artificial intelligence architectures, graphical code generation tools and automatic programming created a certain amount of optimism amongst researchers that they are getting close to the discovery of the ultimate solution. There however still exist many desperate quests for the silver bullet that would ensure higher quality products at lower cost and with much faster response to market demands.

Since Brooks let the proverbial cat out of the bag, a lot of research had been done by various organisations such as IBM[7], the Software Engineering Institute (SEI)[8] and the Standish Group. The SEI focused on analysing and grading larger government based projects while the Standish Group focused more on commercial smaller scale projects. The results from the Chaos Report[9] were sampled on projects limited to six months and six people. These survey results show a steady increase in the success rate of IT projects. However over the last decade the measure for success has also changed from *delivery focused* to customer *satisfaction focused*. The same changes are evident in the changes to the various internationally accepted quality standards such as from International Organization for Standardization (ISO). The Standish Group determined what factors would influence a project to succeed and published their findings in the Chaos Report. The results indicate the following factors in descending order of priority:

- Executive support;
- User involvement;
- Experienced project managers;
- Clear business objectives;

[6] Brooks, Frederick P. 1987. No Silver Bullet. Essence and Accidents of Software Engineering.

[7] International Business Machines (IBM).

[8] The US Department of Defence (DoD) initiated the establishment of the Software Engineering Institute (SEI) that is part of Carnegie Mellon University (CMU).

[9] Standish Group. 2001. The Chaos Report. The Standish Group International.

- Minimised scope;
- Standard software infrastructures;
- Firm basic requirements;
- Formal methodology; and
- Reliable estimates.

These results indicate that the more difficult items such as *reliable estimates* and *formal methodologies* therefore appear to be not that important. Arguably it is not difficult to get users involved and it is not difficult to provide executive support that would positively influence the progress on projects. These factors are however not completely independent and normalised. They have an inherent precedence.

Kast and Rosenzweig[10] emphasised the need to study organisations and management, as a pervasive part of existence that directly and indirectly affects society as a whole. They imply that increased knowledge will somehow lead to better organisation and management. The argument is valid, but the challenge is how to influence and measure performance. Kast and Rosenzweig proposed the following equation:

$$Performance = f(ability, motivation) \qquad \textbf{1-1}$$

Inducing the findings of the Chaos Report into this equation, results in the profound implication that there is a need to augment the abilities and motivation of management and customers. That does not make sense since most of the current emphasis is internally focused on improving the development processes and teams. The problem is assumed to be isolated to the systems development departments while management and the customers have suffered as the victims for many decades. In the Chaos Report this phenomena appears to be *vu ja de*[11].

Ability is comprised of human and technical capabilities that provide an indication of the range of possible performance. Just how much of that latent capability is realised depends on the degree to which individuals and groups are motivated to perform. For organisations, performance results from the aggregation of individual and group efforts to achieve relevant goals.

[10] Kast, F. E., Rosenzweig, J. E. 1970. Organization and Management – A systems approach.

[11] Karl Weick coined the phrase, *vu da je*, to describe the practice of *seeing old things in new ways*. Sutton, R. I. 2002. Weird Ideas That Work – 11½ practices for promoting, managing and sustaining innovation:11.

Measuring and evaluating results is important in determining performance. Output per work-hour, market-share and net profits are relatively straightforward indicators. However, most organisations have multiple goals, some of which are not easily measured. Examples might be customer satisfaction or sustainability. It is important to recognise multiple goals and evaluate organisational performance on a variety of relevant dimensions. It is particularly important to identify substantive functions that spell success or failure in order to give priority attention to them.

Pfeffer and Sutton[12] question why so many managers say so many smart things about how to achieve performance, and work so hard, yet are trapped in firms that do so many things they know will undermine performance.

Philippe Kruchten[13] decrees that a grander vision for software design is required. The process of designing software must be made to fit better with the surrounding engineering processes. There is still a wide gap between users' needs and the way users express requirements on one hand, and the way developers design on the other. The various Standish Group reports make it clear that the primary cause of failure is the inability to deal correctly with users and their changing needs. Developers still struggle to analyse designs, to demonstrate that they are correct and that they fulfil the requirements. Furthermore, there is still a gap between the designs and the code that the programmer fills manually. All these gaps have become narrower in the last 15 years, but they are still major obstacles to consistently producing great products.

Kruchten maintains that design is and should be practiced much broader that what is currently believed. Developers are continuously making design change decisions about the system under construction. They design when they elicit and capture requirements, when they program and test, and finally they design for deployment and disposal. He postulates that software development in a more general framework of engineering design. Software design is therefore a more integrated and more encompassing process. Kruchten concludes that although the silver bullet is still elusive, clear progress in being made. It is establishing foundations with current knowledge and exploring new avenues. The other engineering disciplines have not found a silver bullet, either.

[12] Jeffrey Pfeffer and Robert I. Sutton, 2000. The Knowing-Doing Gap.

[13] Kruchten, P. 2005. Software Design in a Postmodern Era.

1.4.1 The Software Developer's Dilemma

Software development is possibly the most challenging and ambitious career of any. Application domains are limitless, including such diverse fields as arts, biology, economics, management, and weather forecasting. To this end it has been postulated that *all* science *is* computer science.[14] The more accurate disclosure here is that the computer is merely a tool for general science.[15] Each one of these various sciences is in itself complex and its practitioners look towards information science to make it simpler to comprehend and more productive.

To provide information processing solutions, software scientists need to become familiar with these diverse domain areas. They need to acquire *essential knowledge* without necessarily receiving formal education in any particular field of application. The domain experts that commission these projects define the problem and expect the software developers to find the ultimate solution. Software developers are in general very smart people, but they are not omniscient. It is plausible that due to the augmentation qualities of computer technologies these scientists have the potential to solve problems that are not solvable without the use of software. In this sense computer science has an autopoietic quality that *encourages impossible thinking*[16] and hence the software developer becomes the epitome of the knowledge worker or symbolic analyst. Software developers are codifying an artificial virtual representation of the universe as scientists decode the universe.[17] The need for software developers will never seize because as Bill Bryson declares in A Short History of Nearly Everything,[18] now that the human species know what they know they have come to realise that they still do not know that much now and the more they scratch the more they discover and the more there is to scratch. It is an eternal golden braid.[19] Glass and Vessey[20] argues however that most computing researchers are masters of only one or a few domains and therefore do not have the breadth of experience necessary to know what is needed by the many varied domains whose problems are now being addressed. This dilemma forces

[14] Johnson, George. 2001. All Science is Computer Science.

[15] There is no guarantee that the silicon semiconductor based computers as it is known today will remain the ultimate information processing tool.

[16] Wind, Crook & Gunther. 2005. The Power of Impossible Thinking.

[17] Castells, M. 2000. The Rise of the Network Society. Creating a real virtuality.

[18] Bryson, B. 2003. A Short History of Nearly Everything.

[19] Hofstadter, D.R. 1979. Godel, Escher, Bach – An eternal golden braid. Hofstadter's dialogues portray this incumbent complexity very well.

[20] Glass, R.L., Vessey, I. 1998. Focusing on the Application Domain – Everyone Agrees It's Vital, But Who's Doing Anything About It? IEEE Computer.

software developers to become desperately dependant on external help from users, domain experts, business analysts and management consultants. They need to have the aptitude to partner with and seamlessly communicate with all the stakeholders.

1.4.2 Facts and fallacies

Software practitioners are asking software researchers for better advice on how and when to use certain methodologies. The broader programming community are tired of hearing: "Use the latest methodology out of the research labs."[21] The promise of the one-size-fits-all[22] universal solution that solves all of then problems is simply unattainable.[23]

Traditional development methodologies are treated primarily as a necessary fiction to present an image of control. Alternative approaches that recognise the particular *character* of work are required.[24] Robert Glass[25] warns that prescriptive information systems methodologies are unlikely to cope and one should use approaches tailored to each project. Universal, project-independent methodologies are characterised as *weak* in the field of problem solving, while solution approaches focused on the problem at hand are considered *strong*. Glass mocks that there seems to be believed that a universal elixir of some kind is right around the corner, or even presently at hand, or in the latest concept to emerge from research. Coincidently, at the time of Glass' article in 2004, Coburn[26] and Larman[27] have already completed their work to define the realm of applicability of the available methodologies, albeit neither considered the entire spectrum and predominantly focus on the Agile domain. Glass however acknowledges that some of the work done on Agile Development Methodologies and Aspect Oriented Programming is moving in the right direction. It is not enough to propose a new methodology without discussing when its use might be appropriate, and until that happens, it will still be a discipline pretending to cover more than it really does. Glass declared the following generally accepted facts and fallacies of software development.

Glass' facts:

- The most important factor in software development is the quality of the programmers.

---

[21] Glass, Robert L. 2004. Matching Methodology to Problem Domain.

[22] Bereit, Mark. 2006. Escape the software development paradigm trap.

[23] Cooper, M. 2001. Everyone is wrong.

[24] Nandhakumar, J., Avison, D.E. 1999. The fiction of methodological development. A held study of information systems development.

[25] Glass, Robert L. 2004. Matching Methodology to Problem Domain.

[26] Cockburn, A. 2003. People and Methodologies in Software Development.

[27] Larman, C. 2004. Agile & Iterative Development – A Manager's Guide.

- The best programmers are up to 28 times better than the worst.
- Adding people to a late project makes it later.
- One of the most common causes of runaway projects is poor estimation.
- The other most common cause of runaway projects is unstable requirements.
- Requirements errors are the most expensive to fix during production.
- Maintenance typically consumes 40 to 80 percent of software costs.
- Enhancements represent roughly 60 percent of maintenance costs.

Glass' fallacies:

- Software needs more methodologies.
- You teach people how to program by showing them how to write programs.

Even though an increasing number of large projects have been successful compared to similar projects done in the 1980s, very large projects above 10,000 function points in size, missed delivery dates, cost overruns, and outright terminations remain distressingly high even in 2006. The industry is improving, but much more improvement is needed.[28]

1.4.3 Project delivery performance problems

Cockburn[29] spent more than a decade studying what the secrets are to successful software development in the real software business world. He did so by interviewing project teams, participating directly on projects, and reviewing proposals and case studies. Cockburn's research addressed three questions relating to people and software development methodologies.

- Do developers need yet another software development methodology, or can they expect a convergence and reduction at some point in time?
- If convergence, what must be the characteristics of the converged methodology? If no convergence, how can project teams deal with the growing number of methodologies?
- How does the methodology relate to the people on the project?

[28] Jones, C. 2006. Social and Technical Reasons for Software Project Failures.

[29] Cockburn, Alastair. 2003. People and Methodologies in Software Development.

Perkins[30] asserts that the cause of project failures is knowledge. Either managers do not have the necessary knowledge, or they do not properly apply the knowledge they have. If project performance is not as desired, even after consistent application of the project management principle, the underlying principle should be analysed to determine the reason for the continual shortfall. Perhaps the principle is not as sound as some would have you believe.

Evans[31] persists that no matter all the early warning and desperate cries, many companies are still facing some serious project delivery performance problems. These are the standing problems and risks that projects are still facing today.

1.4.3.1 Poor requirements capture

Capturing requirements is arguably the most critical aspect of any project. Errors in the requirements definitions are devastating and are often exacerbated by the following realities:

- Individual business stakeholders are anxious to incorporate *all* of their known requirements into the first or next release.
- Analysts generate hundreds of detailed requirements that often bear little relationship to the business problems that needs to be addressed.
- Most if not all requirements are given a high priority.
- The requirements themselves, at best, represent today's view, which will certainly have changed by the time the requirements are actually implemented.

1.4.3.2 Disconnected design

Given the sheer number of requirements, the design community finds itself spending most of its time trying to figure out what they mean. Meanwhile:

- The requirements analysts move on to other projects, taking with them important tacit knowledge.
- Some stakeholders become concerned that their requirements are not being adequately addressed, and therefore refuse to sign off the designs.
- Other stakeholders unearth more requirements or raise change requests, diverting scarce design expertise onto impact analyses.

[30] Perkins, T.K. 2006. Knowledge:The Core Problem of Project Failure.
[31] Evans, Ian. 2006. Agile Delivery at British Telecom:19.

1.4.3.3 Development squeeze

With the design stage having slipped, development teams find themselves under intense pressure to deliver components into the integration environment by the originally agreed date. In fact, they often take the decision, reluctantly, to start development against an unstable design, rather than do nothing or divert resources to other programmes. Inevitably, system testing is cut short so that original timescales are met and the programme is seen to be on target.

1.4.3.4 The integration headache

The integration team usually has a predefined period during which it needs to integrate what it expects to be fully functional and stable code base. However, due to the instability of the code, and the lack of predefined regression tests, effort is instead diverted to trying to resolve elementary bugs in the delivered code, liaising with a development team that is now engaged in the next major release. Actual integration therefore runs into months, creating a knock-on effect on other programmes requiring the services of the integration team, not to mention frustrations within the business community who had been busy preparing themselves for an on-time delivery.

1.4.3.5 The deployment nightmare

The lapsed time since the requirements was defined and the solution needed nominally extends to anything between 6 and 18 months. Compromises and oversights made during the requirements and design phases, followed by de-scoping during development have resulted in a solution that bears little relationship with what was originally envisaged. Besides, the world has actually moved on in the meantime. The business then finds that the solution is not fit-for-purpose and refuses to adopt it. Worse, they adopt it and soon find that it is slow, error-prone and lacks key features, and eventually revert to the old system.

1.4.3.6 Herding cats in a quagmire

Unequivocally, people are the *original ingredient* necessary for success. However, getting consensus on critical issues is often a political drama that requires parental perseverance. Practitioners and their managers are easily trapped in a destructive tornado of despair caused by autonomic arousal. These complex unexpected events can be fatal if it is not instantly calmed.

1.4.4 Uncontrolled versus uncontrollable processes

Phillip Su[32] has managed developer teams in the Microsoft Windows development group for five years, and has done some interesting reflection on his experience after a recent switch of teams. Making the claim that Windows Vista could be the largest concerted software project in human history, he argues that there is a critical difference between the project being *uncontrollable* opposed to simply being *uncontrolled*. People say that the code is way too *complicated*, and that the pace of coding has been tremendously slowed down by *overbearing process*. There are *cultural barriers hindering truthful disclosure* when reporting schedule slippage. He was interested in the emergent characteristics of failure to deliver. After some discussion with other leaders that have left the group, he jotted down some points on his blog:

> It's certainly true in some sense that they genuinely want to know [the truth]. But in a very important other sense, in a sense that you'll come to regret night after night if you get it wrong, there's really only one answer you can give. After months of hearing of how a certain influential team was going to cause the release to slip, I, full of abstract self-righteous misgivings as a stockholder, had at last the chance to speak with two of the team's key managers, asking them how they could be so … ignorant as to proper estimation of software schedules. Turns out they're actually great project managers. They knew months in advance that the schedule would never work. So they told their VP. And he, possibly influenced by one too many instances where engineering re-routes power to the warp core, thus completing the heretofore impossible six-hour task in a mere three, summarily sent the managers back to "figure out how to make it work." The managers re-estimated … and still did not have a schedule that fit. The VP was not pleased. "You're smart people. Find a way!" This went back and forth for weeks, whereupon the intrepid managers finally understood how to get past the dilemma. They simply stopped telling the truth. "Sure, everything fits. We cut and cut, and here we are. You got it, boss." Every once in a while, Truth still pipes up in meetings. When this happens, more often than not, Truth is simply bent over an authoritative knee and soundly spanked into silence.

This is a common dilemma highlighting the irony in software development management practices at large.

[32] Su, Phillip. 2006. The World As Best As I Remember It.

1.5 Conclusion

The software development industry remains littered with problems relating to delivering products late and missing user expectations regarding functional and quality requirements. Causes are thrown between project managers, analysts, architects, programmers, testers, customers and executive management. The attention is directed towards the various stakeholders whilst little emphasis is placed on technology limitations. Hence several decades of searching for the silver bullet resulted in nothing of significance. What management technique can be applied to a process that is uncontrollable? Does the answer lie hidden in management science, social science, pure science, or computer science? How did the Internet help? Will the next major technological revolution make things better?

Chapter 2
Overview of SDLC methodologies

Traditional scientific method has always been at the very best 20-20 hindsight.
It's good for seeing where you've been.
It's good for testing the truth of what you think you know,
but it can't tell you where you ought to go.
Robert M. Pirsig, 1974

This chapter provides an overview of the SDLC methodology landscape with the aim to define the reference disciplines and characteristics of a varied set of models and methodologies. The text therefore serves two primary causes. Firstly, it is a reference framework for retrospective sensemaking or *hindsight*. Secondly, it is a departure point for ongoing and future endeavours in this field of research.

2.1 Introduction

As introduced in the previous chapter, development lifecycles' started with the origin of life itself in an autopoietic quest for survival through endless cycles of cause-and-effect. This quest developed in as far that man started inventing tools to aid in his survival and earned him the distinction as an intelligent being. However it was not only physical tools. Man also acquired social tools such as complex biokinetical, audible and visual expressions. These were the ancient beginnings of complex adaptive organisational, communications, learning and control systems.[33] These ancient tacit characteristics are encoded into physiological and cultural fabric. Explicit scientific knowledge is relatively young since formally documented systems development knowledge started in the industrial age.

The term *methodology* is more commonly used today while *SDLC* appears to have become an old-fashioned idea. Literature typically uses one or the other. While the term *methodology* could be incorrectly interpreted as too prescriptive, it does however represent a discipline expressed by a set of principles and strategies for analysis and the methods for teaching the

[33] Stacey, R.D. 2001. Complex Responsive Processes in Organizations – Learning and knowledge creation.

discipline itself.[34] The use of the term originated in consulting firms and its interpretation leans towards a holistic and reflective behaviour.[35] The phrase *Agile Methodology* is widely dispersed in literature. It is however not a specific methodology in its own, but instead represents an entire family of SDLC methodologies that *obey* the principles of the Agile Alliance Manifesto.[36]

Most of the recent developments in SDLC methodologies focus on the software development domain.[37] The thesis research reaches beyond the software development domain in order to acquire holistic insights that may have otherwise been overlooked. Conversely, other domains may benefit greatly from the accelerated progress in the software development domain.[38] Imposing a limited view of what lies in the problem domain would create an environment in which it is difficult to serve customers more effectively.[39]

2.2 Industrial Age

In 1776 with his seminal paper titled, *An Inquiry into the Nature and Causes of the Wealth of Nations,* Adam Smith recognised that when a firm is organised around processes based on the specialised content of knowledge, it will gain efficiencies in *producing physical product.* In 1890 Alfred Marshall, in *Principles of Economics*, wrote that knowledge is the most powerful *engine of production.* In 1911 Frederick Winston Taylor published *The Principles of Scientific Management* wherein he created a licence for the *knowledge worker* by writing that managers assume the burden of gathering together all of the *traditional knowledge* which in the past has been possessed by the workmen, and then by classifying, tabulating, and reducing this knowledge to *rules, laws, and formulae which are immensely helpful to the workmen in doing their daily work.*[40]

Shapiro[41] reasons that before *Taylorism*, organisations mostly relied on the workers' initiative to come up with *better methods,* and that this way of doing things were flawed in several respects. It *failed to systematise the improvements* and there was no organisational

[34] Derived from Random House Webster's College Dictionary 2000.

[35] Cockburn, Alastair. 2003. People and Methodologies in Software Development.

[36] This topic is expanded in detail in the second chapter of the thesis.

[37] This reason for this change in focus will be explored in the second chapter of the thesis.

[38] Baskerville and Dove has done significant research and work in extending software based methodologies and applying it to other fields. Baskerville, R.L. Myers, M.D. 2002. IS as a Reference Discipline. Dove, R. 2006. Engineering Agile Systems: Creative-Guidance Frameworks for Requirements and Design.

[39] Frame, J. Davidson. 2002. The New Project Management.

[40] Truch, E. 2004. Knowledge Orientation in Organizations:1.

[41] Shapiro, S. 2002. The Evolution of Innovation. The 24/7 Innovation Group.

learning. Taylor did away with reliance on private knowledge and self-training. He redefined the role of management as follows:[42]

- Develop a science for each element of every person's work, which replaces the old rule-of-thumb method;
- Scientifically select and then train, teach and develop the workers, whereas in the past they chose their own work and trained themselves as best they could;
- Cooperate with the workers to ensure that all the work would be done in accordance with scientific principles; and
- Divide responsibility between management and workers, but management will take over all functions for which they are better at doing than the workers.

Taylor realised that by rationalising processes it is possible to optimise production. The assumption and constraint being that *the processes are completely defined* and therefore *repeatable*. The role of the worker defined as a production engine and that of the manager being defined as *the empirical controller*, monitoring output and taking corrective action on unwanted variations as illustrated in Figure 2-1.

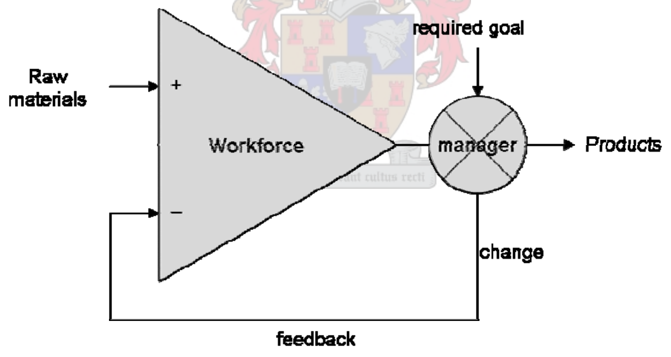


Figure 2-1: Cybernetic model of a manager as empirical controller

Building on the work of Smith, Marshall and Taylor, the Frenchman Henri Fayol defined his fourteen core principles of general management theory in 1916. Fayol's principles however included elements of fair labour practice and gave recognition to the dependency of initiative at all levels of the organisation. These principles have relevance later in the thesis and are therefore enumerated below: [43]

[42] Kast, F. E., Rosenzweig, J. E. 1979. Organization and Management 3rd Edition – A systems and contingency approach:56.

[43] Kast, F. E., Rosenzweig, J. E. 1979. Organization and Management 3rd Edition – A systems and contingency approach:60.

- *Division of work.* Specialisation of labour in order to concentrate activities for more efficiency.
- *Authority and responsibility.* The right to give orders and the power to exact obedience.
- *Discipline.* Essential for the smooth running of business.
- *Unity of command.* Receive orders from one superior only.
- *Unity of direction.* One head and one plan for a group of activities having the same objectives.
- *Subordination of individual interests to general interests.* The interest of one employee or a group should not prevail over that of the organisation.
- *Remuneration.* Fairness and satisfaction for both personnel and the firm.
- *Centralisation.* A natural consequence of organising.
- *Scalar chain.* The chain of superiors ranging from the ultimate authority to the lowest rank.
- *Order.* Provide an orderly place for every individual.
- *Equity.* Fairness and justice pervade the organisation.
- *Stability of tenure.* Time is needed for the employees to adapt to their work and to perform it effectively.
- *Initiative.* Zeal and energy are augmented by initiative at all levels of the organisation.
- *Esprit de corps.* Emphasise the need for teamwork.

Leaping forward in time through two major world wars during which enormous accelerated progress was made in science and technology at a mammoth price, society arrives at a point where they have developed numerous formal methods for systems development that is grounded in the pursuit of military dominance. According to Kast and Rosenzweig,[44] these formal methods flow from a systemic analysis and recognition of a *natural order of activities* during the life cycle of complicated systems. The classical development phases as then

[44] Kast, F. E., Rosenzweig, J. E. 1970. Organization and Management – A systems approach:460-465.

defined are formulated around the external interfaces between government departments and private sector suppliers. The phases are:

- *Conceptual Phase*. Conceiving the proposed solution that will satisfy the mission requirements and objectives.
- *Definitions Phase*. Identifying all the elements or subsystems to be integrated.
- *Acquisition Phase*. Detailed development, production and testing of subsystems.
- *Operational Phase*. Delivering and supporting the system in is operational environment.

The focus was on developing large scale systems that involved multiple parties. Each of these four top-level phases unfolds into sub-phases such as for example in the definitions phase would have: a) request for information, b) request for proposals, c) evaluation and contracting. These classical phases had rigid *gates* whereby the next phase could not start unless the previous phase is completed and signed-off by the acquisition committee or system owner. During the USA ballistic missile programs in the 1950s the *need for urgency* resulted in the emergence of *concurrent engineering*. Each subsystem had to undergo its own systems development life cycle resulting in a *recursive divide-and-concur* approach to systems development. This in turn leads to higher complexity as it introduces more interfaces, more subcontractors, and hence more external influences and interdependencies. For concurrent engineering to succeed, it requires excellent programme management to facilitate timely communication, coordinate activities, configuration and resources amongst diverse engineering teams. Emphasis was on the ramp-up stages that preceded the actual development and commissioning of the systems. It made sense to afford care and attention to details during the analysis and design of these systems. For example NASA's adaptation of the model had three definition phases and the last phase was for the actual development and operations.

It is not possible to anticipate all risks on paper models alone. The Apollo program applied eleven incremental development cycles in order to put the first *man on the moon*. Ten major learning cycles each building on the experience of the former. Within each cycle was four primary phases and within each phase was several sub-phases within which several planned scale models and prototypes were build. Multiple Apollo programmes ran concurrently, overlapping at different stages of its lifecycles. Lessons learned were fed forward into subsequent programmes *without holding up the current launch schedules*. The finer details and in many cases the larger details only emerged when the scientists and engineers *saw what they were thinking*.[45] The end-users, the Astronauts themselves, were highly involved during all the phases. USA was racing the USSR to the moon and hence their model also had to comply with the concurrency requirements forced by this *sense of urgency*. In essence NASA invented very practical SDLC model in the 60s that had many winning attributes that many modern approaches lack. This was *extreme engineering* delivering high-tech high-reliability real rocket science. The engineering was groundbreaking albeit effective and reasonably economical. The *motivation* was *clear and simple*.[46] Kennedy promised the world that America would put a man on the moon before 1970. The *vision* was *clear and simple*.[47] They had a common enemy.[48] The engineers had to devise the *best means possible to win the challenge*. The product was not any of the numerous vessels designed and built, but the end goal of putting man on the moon and bringing him back to earth safely, and by doing so *delighting millions of people*. The Space Programme was specifically interesting in that its success depended on a wide variety of fields of practice ranging from medical science through to the civil engineering used to lay the gravel-stone roads to transport the huge rocket to its launch-pad. The thousands of people that formed part of that programme were fortunate to have had the opportunity to be involved in, and contribute to such a unique project. Many other application domains also depend on a unique and diverse mix of science and engineering, but the common most demanded field is software engineering. The fatal accident of Apollo 1 caused NASA to get ready for the unexpected. They trained themselves to *expect the unexpected* and what to do when the unexpected happens. This is what High Reliability Organisations need to do to survive.[49] So when things suddenly started going wrong during the Apollo 13 mission, the team was ready to take action and *to improvise*. It was a collective effort involving the entire mission support crew to *remote their instincts* through the three Astronauts. They *successfully executed actions that were never planned*. They had to improvise to conserve power and to filter carbon dioxide using a contraption that looked like a primary school art project. This was *astronomical cybernetics*.

[45] Weick, K.E. 1995. Sensemaking in Organizations.

[46] Motivation is provided by a common enemy, an impossible vision and is driven by passion and the need to win.

[47] Purpose is defined by the mission objectives and milestones that together form the winning strategy.

[48] Immelman, R. 2003. Great Boss Dead Boss – How to exact the very best performance from your company and not get crucified in the process.

[49] Weick, K. E., Sutcliffe, K. M. 2001. Managing the Unexpected.

Many budgets were overrun and cost controls had to be tightened up leading to new cost and schedule control mechanisms such as the popular Stage-Gate model.

2.2.1 Stage-Gate process model

Escalating commitment causes management to support projects long after its net value has turned negative. The cost of pushing bad projects forward can be very high. To help avoid this, many managers and researchers suggest implementing tough go/kill decision points in the product development process. The most widely known development model incorporating such go/kill points is the stage-gate process developed by Robert G. Cooper. The stage-gate process provides a blueprint for moving projects through different stages of development.[50]

At each stage, a cross-functional team of people, led by a team leader, undertakes parallel activities designed to drive down the risk of a development project. At each stage of the process, the team is required to gather vital technical, market, and financial information to use in the decision to move the project forward, abandon the project, hold, or recycle the project.

The steps between these points can be viewed as a dynamic process. Stage-Gate divides this process into a series of activities (stages) and decision points (gates).

Stages are:

- *Where the action occurs.* The project team completes key activities to advance the project to the next gate.
- *Cross-functional.* There is no R&D or marketing stage, and each activity is undertaken in parallel to accelerate speed.
- *Where risk is managed.* Vital information is gathered from technical, market, financial, operations to manage risk.
- *Incremental.* Each stage costs more than the preceding one resulting in incremental commitments. As uncertainties decrease, expenditures are allowed to rise and risk is managed.

Gates are:

- Where the Go or Kill decision are taken and prioritization decisions are made.

[50] Schilling, M. A. 2005. Strategic Management of Technological Innovation.

- Where mediocre projects or tasks are culled out and resources are allocated to the best projects.
- Focused on three key issues: quality of execution; business rationale; and the quality of the action plan.
- Where scorecards and criteria are used to evaluate the project's potential for success.

The Stage-Gate approach is very popular with New Product Development groups. However it requires active participations from all the parties and a high management involvement at the gates. Too often the unavailability of the key stakeholders results in delays at gates and consequent stages are held up. IDEO[51] has developed a more agile methodology.

2.2.2 New Product Development

Schilling[52] emphasises that despite its large size, IDEO holds vigorously to its informal culture that encourages playfulness, experimentation, and unfettered creativity. The company's motto is '*fail often to succeed sooner*,' encouraging employees to use brainstorming to rapidly generate ideas. The IDEO management believed in an *innovation funnel approach* whereby a project would start with many ideas that would gradually be filtered down to a single product design. Management also encouraged rapid and frequent *prototyping*, believing that visual aids greatly enhanced the creativity process and facilitated the transmission of ideas. Typically, prototypes were not carefully crafted, functional devices, but instead were rough models quickly put together with such materials as cardboard and foam that could be revised or scrapped as new ideas emerged. Though the company emphasised a free-flowing approach to the innovation process, it also utilised a five-phase system to provide structure to the development process. These phases are:

- *Understand/Observe*. Understand the new client and its business; research everything about previous product models, cost structures and insights about the users and its market.
- *Visualize/Realize*. Prototype models are created to visualize the direction a product solution was heading; maintain close coordination with the client to ensure timely feedback; outline the manufacturing strategy.

[51] IDEO helps organisations innovate through design. Independently ranked by global business leaders as one of the world's most innovative companies, IDEO uses design thinking to help clients navigate the speed, complexity, and opportunity areas.

[52] Schilling, M. A. 2005. Strategic Management of Technological Innovation.

- *Evaluate/Refine*. Build fully functional prototypes to identify and resolve technical problems and issues in the ways users interact with the product; the emphasis shifts from human factors to engineering; complete product design with technical specifications.
- *Engineering*. Verify the manufacturability and performance of the completed product; engineers stayed in close contact with design team members; begin selecting vendors.
- *Implement*. Coordinated release of the product design to the manufacturer; supervise production of tooling, regulatory testing and approvals, and pilot runs of the manufacturing process.

IDEO's process is fit for tangible products of the Industrial Age, but may not be suited for software products and services. A historical overview of information science will aid in the interpretation and understanding of the domain, before delving deeper into more details of SDLC methodologies for the Information Age.

2.3 Information Age

Information science as it is known today is entirely based on *Logic*[53]. On a more practical level its implementation is based on a simple switch that is either on or off. Logic is discrete and each state of the switch leads to a different consequence. This is the simplest possible digital cause-and-effect model that was further developed by Charles Babbage with the Difference Engine in 1822. Based on a designed production function and given stochastic input, the machine would produce output that makes a difference that yields information.[54] In 1847 George Boole derived the mathematics of logic that made it possible to program machines to derive logical conclusions and ultimately make decisions. This consequence gave rise to the possibility of creating Artificial Intelligence (AI)[55] and robots[56] that could one day rule the human race. The Boolean switch became solid-state by means of the

[53] Logic, from the ancient Greek word *logos*, is a field of philosophy rooted in truth theory that is used for the analysis of inference. Today, Boolean Logic is a common term in electronic and information engineering, which developed a collection of Logic building blocks from which inference can be made electronically throw simple transistorised switches.

[54] Bateson, G. 1973. Steps to an ecology of mind:5. Information consists of the differences that make a difference.

[55] In 1950 Alan Turing predicted intelligent machines within 50 years and defined the famous Turing Test.

[56] Isaac Asimov, a biochemical expert and famous science fiction writer devices the universal laws of robotics in 1942.

transistor and the advent of the microprocessor silicon chip[57]. Moore's law[58] was sustained as a result of computing power doubling every 18 months to two years. In 1948 Claude Shannon published the mathematical laws of reliable binary encoded data transmission, thereby giving birth to information networks that earned him the title of *Father of Information Theory*. Engineers defined a few basic primitive machine instructions to soft-wire logic processors[59] to make sense of input-data and produce results. Computers with software are nothing more than a production facility – a means to *get things done*. In essence humans can *transfer* and *defer* some of their capacity to act to an intelligent machine. People use computers as an extension of human being, by augmenting[60] their capabilities and competencies using computerised bionic peripherals. In this mode they extend their senses so that they can more easily make sense of situations and take quicker action. These actions are often enacted and communicated via the same or similar bionic peripherals. The Difference Engine rapidly evolved over two centuries but it is essentially a machine embedded with human intelligence and knowledge. The capacity to act is defined using production rules. The Inference Engine[61] acts on new data by applying the memorised production rules to derive higher order facts that are recursively applied to more production rules until decisions are made and actions are taken to satisfy its intended productivity.[62]

Meanwhile the code-and-fix student cult was growing strongly eventually evolving into the invention of the Personal Computer and independent software technology such as MS-DOS[63]. This gave birth to a huge wave of excitement around how computers could augment human productivity and knowledge.[64] This cult in return fuelled government funded AI

[57] Reid, T.R. 2001. The CHIP – How Two Americans Invented the Microchip and Launched a Revolution. William Shockley, Walter Brattain and John Bardeen invented the transistor in 1947 at Bell Labs. Just over a decade later Robert Noyce helped Jack Kilby to build the first microchip at Texas Instruments.

[58] In 1964 Intel's Gordon Moore made this prediction in an article, but it has been refuted and altered numerously. John Markoff noted that Douglas Engelbart made a similar prediction the same year.

[59] Arithmetic Logic Units (ALU)

[60] Vannevar Bush. 1945. As We May Think. Bush defined, MEMEX, his hope that machines would augment human knowledge.

[61] The term Inference Engine is most commonly used in Expert System technology. However, today it is embedded in many commercial productivity tools such as word processors.

[62] Giarratano, J., Riley, G. 1989. Expert Systems – Principles and Programming.

[63] Microsoft Disk Operating System. Bill Gates and Paul Allen made the historically bold move with IBM to keep ownership of the software and sell licenses independently. Their MS-DOS would thereafter be licensed to run on non-IBM computers based on an open blueprint standard for Personal Computers, which in turn is based on the Intel x86 microprocessor family. This cause in turn gave rise to a market place for Independent Software Vendors (ISV).

[64] Markoff, John. 2005. What the Dormouse Said – How the 60s Counterculture Shaped the Personal Computer Industry.

research at MIT[65] and SAIL[66]. Powerful programming languages evolved such as Fortran[67], Lisp[68], C[69], Basic[70] and Pascal[71]. However these languages were not making it easier for developers to create *good enough software*[72] and thereby causing management to put even more focus on the process of developing code. The Independent Software Vendor (ISV) industry was started by rebels that did not want to be controlled, told what to do, or when to do it. This annoyed traditional management who wanted to measure productivity by counting lines of code written per day.

In 1968 Edsger Dijkstra[73] defined the first rule of *Structured Software Development* and in that same year Friedrich Bauer coined the term *Software Engineering*.[74] It is believed that the term Software Engineering implies levels of rigor and proven processes. Many of the best software developers are in fact university dropouts and promote the idea that software development is more of a skilful art that can only be self-taught than a set of formal scientific proofs and rules.

In stark contrast with Moore's Law, Wirth's Law[75] states that Software gets slower faster than hardware gets faster.

Winston W. Royce,[76] Frederick Brooks[77] and Barry Boehm[78] laid the foundation for Software Development practices by respectively addressing the human and systems

[65] Massachusetts Institute of Technology (MIT).

[66] Stanford Artificial Intelligence Laboratory (SAIL).

[67] Formula Translation developed at IBM by John W Backus in the mid 50s.

[68] LISt Processing (LISP) computer programming language invented by John McCarthy's team, at SAIL, in the late 50s.

[69] Programming language invented by Dennis Ritchie, at Bell Labs, in the late 60s.

[70] Beginner's All-purpose Symbolic Instruction Code (BASIC) was invented by John George Kemeny and Thomas Eugene Kurtz, at Dartmouth College, in 1963.

[71] Nicklaus Wirth invented Pascal, at ETH, in the early 70s.

[72] Bach, James. 1995. The Challenge of Good Enough Software. American Programmer Magazine.

[73] Dijkstra, EW. 1968. GOTO Statement Considered Harmful. Communications of the ACM. Vol. 11(3):147-148.

[74] MacKenzie, Donald. 2000. A view from the Sonnenbichl: on the historical sociology of software and system dependability. Proceedings of the international conference on History of computing: software issues. Paderborn, Germany. Springer-Verlag:New York:97-122.

[75] Niklaus Wirth, inventor of Pascal, made this statement in 1995.

[76] Royce, W.R. 1970. Managing the development of large software systems. Proceedings IEEE WESCON.

[77] Brooks, F.P. 1987. No Silver Bullet. Essence and Accidents of Software Engineering Computer. Vol. 20, No. 4.

[78] Barry W. Boehm. 1979. Software engineering-as it is. Proceedings of the 4th international conference on Software engineering. IEEE Press.

perspectives. Brooks challenge managers to give recognition to great designers. Brooks proposed some steps to put *more emphasis on design*:

- Systematically identify top designers as early as possible. The best are often not the most experienced.
- Assign a career mentor to be responsible for the development of the prospect, and carefully keep a career file.
- Devise and maintain a career development plan for each *prospect*, including carefully selected apprenticeships with top designers, episodes of advanced formal education, and short courses, all interspersed with solo-design and technical leadership assignments.
- Provide opportunities for growing designers to interact with and stimulate each other.

Royce applied metaphors such as the Waterfall model and Boehm introduced *iterative* flows between development stages with his Spiral model with multiple *incremental* lifecycles. The eight stage Waterfall model was an attempt to combat the infamous *code-and-fix* or hacker methodology that emerged naturally amongst students and amateurs of the 70s and early 80s.[79] The inherent problem of the Waterfall approach was that not all information was known at earlier planning and specification stages. The requirements were simply not known in enough detail by any of the stakeholders. It was up to the developers to *discover* the details as they go about coding a solution. This lead to the simpler *incremental* Spiral model flowing quickly through all four phases of a smaller initial scope and then repeating the spiralling cycles outwards until the complete system is build. *Do-it-twice* or *throw-away prototyping* evolved and was encouraged by astute masters such as Brooks. Yet, the demand for formality did not fade and an apposing camp was working on structured programming and modelling techniques.

2.3.1 Unifying of Modelling Conventions

In the mid-1980s Ed Yourdon, Tim Lister and Tom DeMarco formalised techniques for Structured Analysis and System Specification.[80] In the early-1990s James Martin started the field of Information Engineering (IE) and spurred the use of Computer Aided Software

[79] Giarratano, J., Riley, G. 1989. Expert Systems – Principles and Programming:360.

[80] Demarco, T. Plauger, PJ. 1979. Structured Analysis and System Specification.

Engineering (CASE) tools while Coad and Yourdon published their books on Object-Oriented Analysis[81] and Design.[82]

This lead to the development of rapid prototyping tools and structured modelling techniques such as ERD[83], IDEF[84], and OMT[85] to help engineers to translate *perceived requirements* into a *visual model* that could be verified and validated against the system requirements *before* coding starts. CASE tools promised to be the *silver bullet* by automatically generating the computer code directly from these visually designed models. During the same time computer hardware advanced at the speed predicted by Moore's law and many computer languages evolved promising easier coding and that object-oriented programming (OOP) would produce quality software products. The Object Management Group (OMG) promoted a vision that any two independently developed software components would be able to interface seamlessly across platforms, systems and networks using CORBA[86] as the industry standard for interoperability. Java[87] made similar promises based on virtual runtime ports. Microsoft developed COM[88] and eventually their .NET framework as the silver bullet based on the concept of *managed code* running inside a robust Common Language Runtime (CLR) kernel.

All of this hype not only caused damage but put far too much *emphasis on the wrong development phase*. Somehow it seems that the Software Engineering industry is fooled into believing that the *code and debug phase* is flawed and that programmers are intentionally developing the wrong products and doing it badly. Hence the SEI developed their five-step CMMI metric for measuring the process maturity of teams. Three of the fathers of Object Oriented Design notations, Grady Booch, James Rumbaugh, and Ivar Jacobson, unified their ideas under the Rational[89] umbrella and dominated the standardisation for UML[90]. Rational then also continued refining the Rational Unified Process (RUP) SDLC model which is being

[81] Coad, P. Yourdon, E. 1990. Object-Oriented Analysis.

[82] Coad, P. Yourdon, E. 1991. Object-Oriented Design.

[83] Entity Relationship Diagrams (ERD).

[84] Integrated DEFinition Methods (IDEF). Level 0 is at the business process level, then iterative analysis decompose the models down to the lowest ontological level 5. See www.idef.com

[85] OMT – Object Modelling Technique.

[86] CORBA – Common Object Request Broker Architecture.

[87] Java is a programming languages invented by Sun Microsystems' James Gosling in 1994.

[88] COM – Component Object Model.

[89] Rational is now owned by IBM Software Group.

[90] Unified Modelling Language (UML).

adopted as a standard Unified Process (UP). RUP is essentially a Spiral model[91] with the following four key phases: Inception, Elaboration, Construction and Transition. Each of these phases incorporates requirements definition, design, implementation and deployment artefacts.

Object-Oriented (OO) concepts in software development had no single origin. It evolved synchronously and rapidly from the late-1960s with Simula-67 and Smalltalk in the early 70s. Smalltalk was the first pure OO development language, but most of the existing programming languages quickly adapted and C++ eventually overtook the Smalltalk market in the late-90s. OO was already very useful for expert, control and real-time systems development where the frameworks and blackboards were already developed around the modelling of real-world objects.

OO provides design and construction mechanisms that are necessary for flexible open-ended systems that can be changed at a component level without requiring complete system refactoring and development. These loose coupled binding allow models to become dynamic and takes advantage of technology evolution that extends interfaces across network boundaries.[92]

In 1992 Ivar Jacobson[93] wrote a book titled Object-Oriented Software Engineering in which he introduced the concept of Use Case modelling that brought a more human oriented systems analysis and design to a largely abstract domain. An equally important concept he brought into software design is to draw conventional wisdom from other more mature industries. Jacobson applied various industry metaphors to software development such as creative design, construction and long-term support. Four years earlier Bertrand Meyer[94] completed his book on a similar theme called Object-Oriented Software Construction, but his approach was from a quality perspective that introduced the concepts of interfaces and contracts to software development.

Formal methods were a prerequisite in financial, military and government institutions. Most of the products developed was generally considered once-offs that took several calendar years and hundreds of man-months to complete. Many projects are often cancelled. Angry stakeholders started to call for industry standards whereby quality can be assured up front.

[91] Boehm, B. 2000. Spiral Development – Experience, Principles, and Refinements. SEI special report.

[92] Baresi, L. Di Nitto, E. Ghezzi, C. 2006. Toward Open-World Software: Issues and Challenges.

[93] Jacobson, I. 1992. Object-Oriented Software Engineering.

[94] Meyer, B. 1988. Object-Oriented Software Construction.

2.4 Standards

Quality standards are important for the protection of consumers as much as it is for economic growth and technological progress. Standards do however have a downside in that it fossilises the capability and creativity for *radical innovation* that leads to enormous value.[95]

Standards are derived *implicitly* from dominant designs and *explicitly* by committees. Some standards are very simple, practical, useful and essential, for example the international ‘Rules of the Road’ for seafaring vessels. Another useful albeit more complicated standard is the OSI 7-Layered Communications Reference Model[96] that made the Internet possible. With seafaring there should be no confusion and the rules are to be obeyed instinctively. Reference models however need to be tailored and developed for its particular use. The seafaring rules need to be obeyed by people whilst the communications protocols need to be adhered to by electronic hardware and software systems.

Crosby[97] recognised that Quality programmes are notoriously difficult to quantify. When an organisation is measuring nothing, the only recourse it knows is to scrap and rework, and often even these statistics are not being tracked effectively. Once a formal system is introduced, much more accurate data starts to emerge and initial costs of quality often appear to increase. The most concise and well-regarded statements of how to achieve quality is W. Edward Deming's 14 principles in his *Theory of Profound Knowledge*. [98]

Useful and enduring standards are those used for measurement and control. In the mid 1980s the US Department of Defence needed a standard measure for determining whether contractors could provide software on time, within budget and to specifications.

2.4.1 ISO 12207

The US Department of Defence (DoD) is a pioneer in defining software development life cycles. The DoD undertook an effort to unify DoD-STD-2167A and MIL-STD-7935 to create one life-cycle standard identified as MIL-STD-498. The policies however shifted toward more reliance on commercial standards. The Institute for Electrical and Electronics Engineers (IEEE) and the Electronics Industry Association (EIA) then initiated a joint project to create a commercial replacement for MIL-STD-498, resulting in single standard titled the

[95] Boisot, M. 1998. Knowledge Assets.

[96] ISO/IEC. 1994. Information technology -- Open Systems Interconnection -- Basic Reference Model: The Basic Model. International Standard 7498-1.

[97] Crosby, P.B. 1979. Quality is Free. McGraw-Hill.

[98] Anderson, David J. 2005. Stretching Agile to fit CMMI Level 3.

IEEE Trial Use Standard 1498 and the EIA Interim Standard 640. The American National Standards Institute (ANSI) designated the document as ANSI Joint Standard 016 (J-016).

International Organization for Standardization (ISO)[99] standard 12207[100] was also underway. ISO 12207 offers a framework for software life-cycle processes from concept through retirement. It is intended for two-party use where an agreement or contract defines the development, maintenance, or operation of a software system. As is conventional to military standards, it uses a language of "shall" to indicate mandatory provisions, "should" for recommendations, and "may" for permissible actions. It provides a structure of processes using mutually accepted terminology, rather than dictating a particular SDLC methodology. Since it is a relatively high-level document, it does not specify the details of how to perform the activities and tasks comprising the processes. Organizations adopting ISO12207, therefore still need to use additional standards to fill in the details.

ISO 12207 describes *five primary processes* – acquisition, supply, development, maintenance, and operation. It then divides the five processes into *activities*, and the activities into *tasks*, while placing requirements upon their execution. It also specifies *eight supporting processes* – documentation, configuration management, quality assurance, verification, validation, joint review, audit, and problem resolution – as well as *four organizational processes* – management, infrastructure, improvement, and training.

In 1992, the Institute of Electrical and Electronics Engineers (IEEE) had completed its own life-cycle process standard 1074, providing detailed descriptions of development and maintenance activities as well as their connections. In principle, one could use IEEE 1074 to construct processes that would comply with the requirements of either J-016 or ISO 12207.

Traditionally these standards require organisations to *tailor* the appropriate processes to fit the scope of their particular projects. Compliance is attained by the performance of the selected processes, activities, and tasks.

2.4.2 ISO 9000

The ISO 9000[101] family of standards listed below has been developed to assist organisations, of all types and sizes, to implement and operate effective quality management systems.

[99] International Organization for Standardization Website http://www.iso.org

[100] Moore, James. 2006. ISO 12207 and Related Software Life-Cycle Standards

[101] ISO. 2000. SABS ISO 9000:2000 edition 2. ISBN 0-626-12810-2

- ISO 9000 describes fundamentals of quality management systems and specifies the terminology for quality management systems.
- ISO 9001 specifies requirements for a quality management system where an organisation needs to demonstrate its ability to provide products that fulfil customer and applicable regulatory requirements and aims to enhance customer satisfaction.
- ISO 9004 provides guidelines that consider both the effectiveness and efficiency of the quality management system. The aim of this standard is improvement of the performance of the organisation and satisfaction of customers and other interested parties.

Together they form a coherent set of quality management system standards thus *facilitating mutual understanding*. These standards promotes that to be successful it is necessary to direct and control processes in a systematic and transparent manner, and that success can result from implementing and maintaining a management system that is designed to continually improve performance while addressing the needs of all interested parties. Managing an organisation encompasses quality management amongst other management disciplines.

ISO 9000:2000 proposes the following quality management principles that can be used by top management in order to lead the organisation towards improved performance:

- *Customer focus*. Understand current and future customer needs, meet customer requirements and strive to exceed customer expectations.
- *Leadership*. Establish unity of purpose and direction for the organisation; create and maintain the internal environment in which people can become fully involved in achieving the organisation's objectives.
- *Involvement of people*. Involvement of people at all levels enables their abilities to be used for the organisation's benefit.
- *Process approach*. Manage activities and related resources as a process.
- *System approach to management*. Identifying, understanding and managing interrelated processes as a system contributes to the organisation's effectiveness and efficiency in achieving its objectives.
- *Continual improvement*. Improve performance on a permanent, continual basis.
- *Factual approach to decision making*. Effective decisions are based on the analysis of data and information.

- *Mutually beneficial supplier relationships*. Interdependent and mutually beneficial relationships enhance the ability to *create value*.

Whereas these standards apply to processes in general, there exist more specific standards for project management processes.

2.4.3 ISO 10006

ISO 10006 gives guidance on the application of *quality management in projects* and according to ISO, it is applicable to projects of varying complexity, small or large, of short or long duration, in different environments, and irrespective of the kind of product or process involved. It necessitates some tailoring of the guidance to suit a particular project. ISO claims that the standard provides guidance on quality system elements, concepts and practices for which the implementation is important to and has an impact on the achievement of quality in project management.

Pither and Duncan[102] however believes that the application of this document is more likely to have the opposite effect, and warns that if attention is given to the items identified in the standard at the expense of others critical to project management, the result could very well be a poorly managed and unnecessarily costly project that is compliant with the standard. They argue further that there is no project execution process. There is however a lot of planning and controlling processes, but no place to actually *do the work* of the project. This omission regrettably reinforces the notion that project management is limited to *planning and controlling*. The standard recognises that project phases and project life cycles exist, but it provides no guidance on how the identified project processes relate to project phases. Work breakdown structure, critical path, project objectives, project life cycle, project network diagram, project scope and many other project management terms are used without being defined. Since many of these terms are understood differently in different application areas, the absence of agreed definitions may cause considerable confusion.

2.4.4 ISO/IEC 15288

Since October 2002 ISO/IEC 15288[103] became the grand new Systems Life Cycle Process standard that embodies ISO 12207. It establishes a common framework for describing the life cycle of systems created by humans. It defines a set of processes and associated terminology. These processes can be applied at any level in the hierarchy of a system's

[102] Pither, R., Duncan, W. R. 2006. ISO 10006 : Risky Business. Project Management Partners.

[103] Magee, S. 2002. ISO/IEC 15288 marketing presentation.

development. ISO/IEC 15288 primarily concerns those systems that are man-made and may be configured with one or more of the following: hardware, software, humans, or processes. In general terms it is a generic life cycle process standard providing:

- A holistic view of software and systems engineering;
- A basis for stage-based life cycle models;
- A process framework that can be tailored to suit its application;
- A framework that reduces development risk; and
- A basis for communicating.

2.4.5 ISO/IEC 19759

ISO/IEC 19759 is a guide to the software engineering body of knowledge (SWEBOK). It identifies and describes that subset of the body of knowledge that is, according to ISO, generally accepted, even though software engineers must be knowledgeable not only in software engineering, but also, of course, in other related disciplines. SWEBOK is an all-inclusive term that describes the sum of knowledge within the profession of software engineering.

2.4.6 CMMI

Carnegie Mellon University's Software Engineering Institute established the Capability Maturity Model as a way to assess and describe the quality of an organisation's software development. First released in 1991, it was largely associated with the government contracting, aerospace and defence industries focused on large, long duration programs of life critical systems with government requirements for auditability and traceability. The model was eventually extended to incorporate 25 process areas and by 2000 many of the disparate elements were brought together into a single framework known as the Capability Maturity Model Integration (CMMI).[104]

CMMI is a way to assess and describe an organisation's software development process, compare it against industry standards and help the organisation refine and improve that process. It combines a carefully chosen set of best practices based on experience in a variety of disciplines, including systems analysis and design, software engineering and management. With CMMI, an organisation can simultaneously tackle a range of improvements that would otherwise be addressed as free-standing initiatives. This, in turn, encourages improvement

[104] Kay, Russell. 2005. CMMI.

throughout the enterprise and helps organisations consider the full product development life cycle.[105]

CMMI provides two basic models: staged and continuous. Staged CMMI is the better known, with its five levels of maturity:

- *Level 1: Initial*. Chaotic, ad hoc, heroic;
- *Level 2: Repeatable*. Project management, process discipline;
- *Level 3: Defined*. Institutionalised, defined & confirmed standard business processes;
- *Level 4: Managed*. Quantified process management and measurement takes place; and
- *Level 5: Optimising*. Predictable, process improvement, deliberate process optimisation.

The model enables comparisons between organisations and offers a proven sequence for improvement. Continuous representation of CMMI lets an organisation select a set of specific improvements that best meet its business objectives and minimise risk, while perhaps making it easier to compare processes across projects and to transition from other quality standards. Within each of the CMMI maturity levels, key processes are defined in five areas: goals, commitment, ability, measurement and verification.

SEI has developed a rigorous instrument, the Standard CMMI Appraisal Method for Process Improvement (SCAMPI), which provides detailed ratings of strengths and weaknesses relative to the CMMI models. SCAMPI helps organisations improve their processes by setting priorities and focusing on improvements that match business goals.[106]

The main difference between CMMI and ISO 9001 is that ISO 9001 specifies a minimal acceptable quality level for software processes, while CMMI establishes a framework for measuring continuous process improvement and is more explicit in defining the means to that end. Due to origin and centre of influence the ISO is favoured in Europe and CMMI in North America. Economic development agencies in India and Ireland, have praised CMMI for allowing them to compete for U.S. outsourcing contracts. This has had a very positive effect on the employment of software engineers in Third World economies, but it has also adversely affected the high-tech job market in developed countries.

[105] Kay, Russell. 2005. CMMI.

[106] Kay, Russell. 2005. CMMI.

Luxoft[107] was the first European software company to achieve a Level 5 CMMI-certification, which means a high-level description of (and control over) software processes. In achieving this certification, Luxoft increased its tool dependence by adopting a number of tools.

CMMI models are not process descriptions and it does not guide organisations on how to implement improvements in software development. It merely indicates where improvements are required. The traits CMMI needs to measure are easy to recognise, but difficult to cultivate and often rare. CMMI however has rigid requirements for documentation, measurement and step-by-step progress. This makes it better suited to large organisations than to small. But even most large ISVs such as Apple and Microsoft rarely manage their requirements documents as formally as CMMI requires. Since documentation is a requirement for Level 2, most ISVs are stuck at Level 1.

To help organisations to climb the CMMI ladder, SEI developed the IDEAL[108] life-cycle model for software process improvement. Recognising that the model had great potential outside of the process arena, the SEI revised the IDEAL Model for broader application.[109]

IDEAL provides a practical approach to continuous improvement by outlining the steps necessary to establish a successful improvement program. Following the phases, activities, and principles of the IDEAL model has proven beneficial in many improvement efforts. The model provides a disciplined engineering approach for improvement, focuses on managing the improvement program, and establishes the foundation for a long-term improvement strategy.

[107] Pries-Heje, J., Baskerville, R. L., Hansen, G. I. 2005. Strategy Models For Enabling Offshore Outsourcing:13.

[108] IDEAL is an acronym derived from the five lifecycle phases which it describes.

[109] Gremba, J. Myers, C. 1997. The IDEAL Model: A Practical Guide for Improvement.

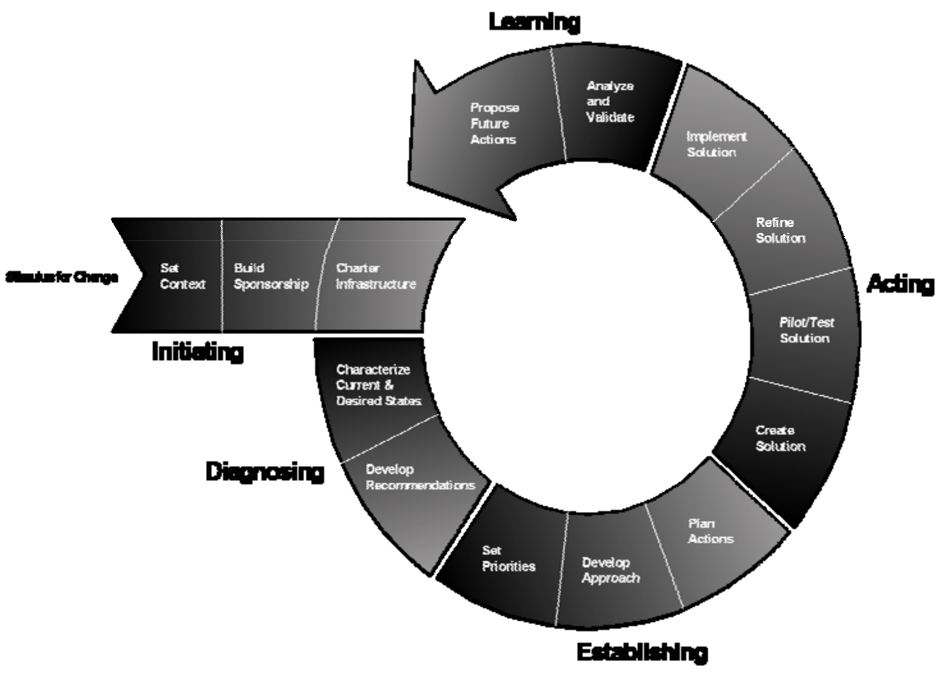


Figure 2-2: SEI IDEAL Lifecycle Model[110]

The IDEAL lifecycle model shown in Figure 2-2 consists of five phases from where the acronym is derived:[111]

- ***I**nitiating*. Laying the groundwork for a successful improvement effort.
- ***D**iagnosing*. Determining where you are relative to where you want to be.
- ***E**stablishing*. Planning the specifics of how you will reach your destination.
- ***A**cting*. Doing the work according to the plan.
- ***L**earning*. Learning from the experience and improving your ability to adopt new technologies in the future.

In the Initiating phase the focus is put on the understanding of business goals and objectives. During the diagnosing phase two characterisations of the organisation are developed: the current state and the desired future state. The purpose of the Establishing phase is to deliver a detailed work plan for process improvement. The main activities include: setting priority, developing approach, and planning action. The purpose of the Acting phase is to implement the work that is conceptualised and planned in the previous three phases. The main activities include creating, piloting, testing, refining, and implementing. In the Learning phase, the entire IDEAL experience is reviewed to determine what was accomplished, whether the

[110] Gremba, J. Myers, C. 1997. The IDEAL Model: A Practical Guide for Improvement.
[111] Gremba, J. Myers, C. 1997. The IDEAL Model: A Practical Guide for Improvement.

effort accomplished the intended goals, and how the organisation can implement change more effectively and efficiently in the future. CMMI and IDEAL is grounded in the PDCA[112] principles defined by W. Edwards Deming.

2.5 General overview of the software development process

Paul Bleicher[113] uses a *Home Renovation* metaphor for explaining the software development process. The language of software development derives from the oldest engineering field with phrases such as *designing the architecture* and *building the foundation*. Despite the similarities between construction and software development, the one difference which is important to note is that in construction, the design and testing processes are relatively short compared with the construction itself, while in software engineering this is reversed. The traditional design process is typically quite long and involved compared to the final development stage. *Extreme Programming*[114] challenges this traditional approach and encourages less emphasis on a big upfront design and documentation. Traditional SDLC methodologies are therefore inherently flawed when applied to the software development domain. This traditional approach is detailed in the following paragraphs to define a reference background for the modern methods that will be discussed in the subsequent text.

2.5.1 Requirements analysis and design phase

As with the building of a house the development of software begins with a requirements phase. Those responsible for the development of requirements must listen carefully to real users of the products, understand, and even observe the workflow of the process being automated. Often the end users will not understand how to translate their needs into a systematic requirement, nor will they have fixed ideas about how they want the software to work. The requirement gatherer must sift through much of the information and develop requirements that are specific enough to base the design of a system on, but not too prescriptive. The requirements must be checked and tested with the end-users for feasibility, completeness, ambiguity, and consistency. A final document should be the agreed upon requirements for the software and to be used for checking against the end result. This document is similar to the high-level design drawings of the architect. Although these requirements may change during the software building process, changes come at a cost of time and increased expense. Again, the later in the process a change occurs, the greater the

[112] Plan-Do-Check-Act (PDCA).

[113] Paul Bleicher. 2003. Software Life Cycles and Kitchen Sinks.

[114] Extreme Programming is one of the various Agile software development methodologies which is detailed in paragraph 2.7 on page 50.

cost. Once the requirements are complete, the software team will begin a detailed design specification, much as the construction architect went from high-level drawings to detailed wiring, plumbing, and cabinetry plans. In developing a specification, the software architect makes many decisions about what functionality will be contained in each of the different software modules and how these modules interact. The specification document will form the basis of the test plan for unit and final testing of the software.

2.5.2 Development and implementation phases

Following the agreed upon design, the software will be implemented. The actual software code will be written and the various modules will then be integrated together. Each individual unit or module will be tested, and the overall integrated product will be tested to make sure that it works as specified. Once the project is completed it must be accepted by the end users, often through another series of practical tests. It will then be installed and deployed in production. Over time the software will require continued maintenance during operation, which may be as simple as changing an installation parameter, or as complex as rewriting some of the code. Finally, the software will eventually reach an end of its usable life, at which point it will likely be replaced by a new product. The life span of software may be brief, like that of Web browsers, or very long, as in the case of some COBOL-based banking systems from the 1960's that are still in use today.

2.5.3 Waterfall model

The SDLC model described above is commonly referred to as the Waterfall model. Each step takes input from the previous step and produces output that is used by the next step. The development of the software flows from step to step like a waterfall. The typical steps are:

- Requirements Definition;
- Analysis and design;
- Coding and testing;
- Acceptance testing;
- Installation and deployment; and
- Maintenance and support.

Although the Waterfall model is quite straightforward it doesn't work well in the current world of rapid application development of complex projects. Waterfall development works best when the user starts the process specifying requirements, but isn't involved thereafter until final testing. It is a relatively inflexible model that limits the ability to predict timing of a project and resource requirements. The Waterfall method only works in an environment where change occurs rarely, so that portions or stages of a project can be completed, closed, not reopened. For this reason, a number of other theoretical models have been developed.[115]

2.5.4 Alternatives to the Waterfall model

One commonly used alternative to the Waterfall method is *Rapid Prototyping* as originally proposed by James Martin.[116] Using languages and tools that make it possible to quickly assemble an application, the programmers can develop one or many prototypes to test requirements and specifications. Users interact with the prototypes, suggesting modifications and improvements. When all prototyping is completed the prototypes are set aside and the code is completely rewritten in a more robust environment, often using a version of the Waterfall methodology. A risk of rapid prototyping is that some aspects of the prototype may be too difficult or expensive to build as part of the final system. The user may have to accept less functionality from the final system than the prototyped one.

Prototypes can be illustrative prototypes, which can be nothing more than a mocked-up Web page or even storyboard. Other prototypes are functional, which can actually deliver part or all of a working system or part of a system. A major risk of functional prototypes is the temptation to incorporate the rapidly developed prototype into the final system, thus sacrificing quality and integrity of the process and product. An even bigger risk is that management and end-users perceive the development progress to be further than it actually is.

There are two types of prototypes: throwaway and evolutionary.[117] Throwaway prototypes are built in a quick and dirty manner, are given to the customer for feedback, and are thrown away once the desired information is learned. The desired information is captured in a requirements specification for a full-scale product development. Evolutionary prototypes are built in a quality manner, are given to the customer for feedback, and are modified once the desired information is learned to more closely approximate the needs of the users. This process is repeated until the product converges to the desired product.

[115] Paul Bleicher. 2003. Software Life Cycles and Kitchen Sinks.

[116] James Martin. 1991. Rapid application development. Macmillan Publishing.

[117] Davis, A. 1992. Operational Prototyping: A New Development Approach. IEEE Software. Vol.9. No.5:70-78.

Throwaway prototypes should be built when critical features are poorly understood. Evolutionary prototypes should be built when the critical functions are well understood but many other features are poorly understood. Build a throwaway prototype followed by a from-scratch evolutionary prototype if most functions are poorly understood. When the risks are high it calls for a Concept Evaluation with throwaway bread-board prototyping to tackle the nasty risks and proof concepts are practical and economical. The second type of prototype is not really prototyping but iterative chunks of *real* high quality product development.

The SDLC methodology known as the *Incremental* model assigns requirements to a particular section of the software. The sections are each built to completion and tested individually, beginning with the highest priority module. As each module is added on, the product takes on more and more functionality. Microsoft manages huge projects with a related methodology using a Synchronize and Stabilize method whereby the entire project is built each night from the currently completed modules. Incremental models allow for adaptation to changing requirements through user interaction with the partially completed software.

Another methodology that is related to the Incremental methodology is the *Spiral* model,[118] whereby the modules that have the highest risk inherent in them are prototyped. As these are completed, the risk is reassessed and the next highest risk module is prototyped. As more and more of the software get completed, the process becomes more straightforward and carries much less risk, until the final software is built from the increments.[119] Think of it as having a large rock, some pebbles, sand and water. One needs to get all these ingredients into a jar that has just enough volumetric capacity. One must start with the largest objects and finish by carefully pouring the water into the jar.

There are many more models of SDLC methodologies, including the V model, Sawtooth, Shark's tooth, and others. The most radical to date is Extreme Programming (XP) wherein the software is rapidly built through an iterative process from *user stories* without the need for formal documentation and planning. Programmers work in teams of two on discrete parts of the software, providing rapid and high quality development. This form of SDLC requires a highly committed team with collective ownership and knowledge. XP works well for first-

[118] Boehm, B. 2000. Spiral Development:Experience, Principles,and Refinements.

[119] Paul Bleicher. 2003. Software Life Cycles and Kitchen Sinks.

time products in start-up companies.[120] XP is just one of many recently developed methodologies which falls under a category called Agile Development Methologies.

In simple terms, all of the SDLC models are different variations on making sure that software is well defined at the beginning, properly built, extensively tested, and smoothly implemented. In the end, the use of any SDLC methodology creates a process that ideally is predictable, repeatable, measurable, and efficient. The SDLC provides a high-quality process for the development of high-quality products that can achieve regulatory compliance, in software development and in general the discipline of a fixed process can bring order, but can also suppress innovation, value and sustainability. It is essential that everyone involved consider what they are doing, and for whom, throughout the project. Despite the many pressures to complete the product on time and on budget, each participant must be ready to question, re-examine and modify the process and product to deliver the best possible results.[121]

2.5.5 Evolutionary model

Independent software vendors, such as Microsoft, are focused on making their software better by continuously supporting and improving it to keep pace with user demands. These software products evolved and new feature rich versions were released almost annually to the delight of their customers. Software engineers have traditionally considered any work after initial delivery as simply software maintenance. Some researchers have divided this work into various tasks, including making changes to functionality (perfective), changing the environment (adaptive), correcting errors (corrective), and making improvements to avoid future problems (preventive).

[120] Paul Bleicher. 2003. Software Life Cycles and Kitchen Sinks.

[121] Paul Bleicher. 2003. Software Life Cycles and Kitchen Sinks.

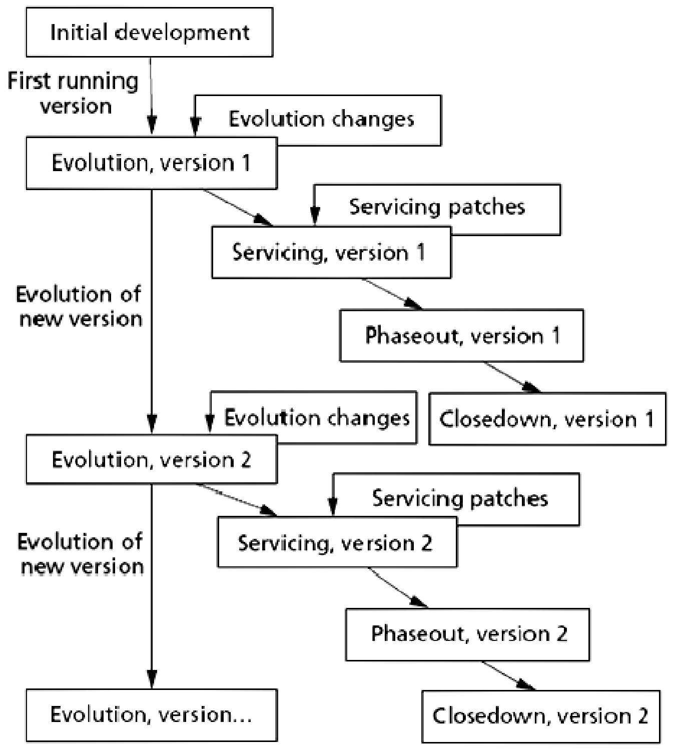


Figure 2-3: Evolutionary Software Lifecycle Model[122]

This gave rise to a versioned staged model as shown in Figure 2-3 by extending the Spiral model into the following stages:

- *Initial development*. Develop the system's first functioning version.
- *Evolution*. Extend the capabilities and functionality of the system to meet user needs, possibly in major ways.
- *Servicing*. Engineers make minor defect repairs and simple functional changes.
- *Phaseout*. Do not undertake any more servicing, seeking to generate revenue from the system as long as possible.
- *Closedown*. Withdraw the system from the market and directs users to a replacement system, if one exists.

Rajlich and Bennett[123] warns that managers should watch for situations in which *software maintenance* is considered one homogeneous phase and handed over to second-rate programmers or outside contractors. Retaining highly skilled staff from initial development

[122] Rajlich, V. T., Bennett, K. H. 2000. A Staged Model for the Software Life Cycle.
[123] Rajlich, V. T., Bennett, K. H. 2000. A Staged Model for the Software Life Cycle.

through evolution is crucial because it is often impossible to codify and make explicit the tacit knowledge of these experts.

2.5.6 Unified Process (UP)

The Unified Process (UP) is a use-case-driven, architecture-centric, iterative and incremental development process framework that leverages the Object Management Group's (OMG) UML and is compliant with the OMG's Software Process Engineering Metamodel (SPEM). The UP is broadly applicable to different types of software systems, including small-scale and large-scale projects having various degrees of managerial and technical complexity, across different application domains and organizational cultures.

The UP emerged as the unification of Rational Software Corporation's Rational Approach and the Objectory process in 1995 when Rational Software Corporation acquired Objectory. Rational Software Corporation developed the Rational Approach as a result of various customer experiences, and Ivar Jacobson created the Objectory process primarily as a result of his experience with Ericsson in Sweden.

Today, the IBM Rational Unified Process (RUP) is a widely adopted, well known, and well-defined system development process. It provides several mechanisms, such as relatively short-term iterations with well-defined goals and go/no-go decision points at the end of each phase, to provide management visibility into the development process.

The UP defines the following four phases:

- *The Inception phase*, concluding with the *Objective milestone*, focuses on establishing the project's scope and vision; that is, establishing the business feasibility of the effort and stabilising the objectives of the project.
- *The Elaboration phase*, concluding with the *Architecture milestone*, focuses on establishing the system's requirements and architecture; that is, establishing the technical feasibility of the effort and stabilising the architecture of the system.
- *The Construction phase*, concluding with the *Initial Operational Capability milestone*, focuses on completing construction or building of the system.
- *The Transition phase*, concluding with the *Product Release milestone*, focuses on completing transitioning or deployment of the system to the user community.

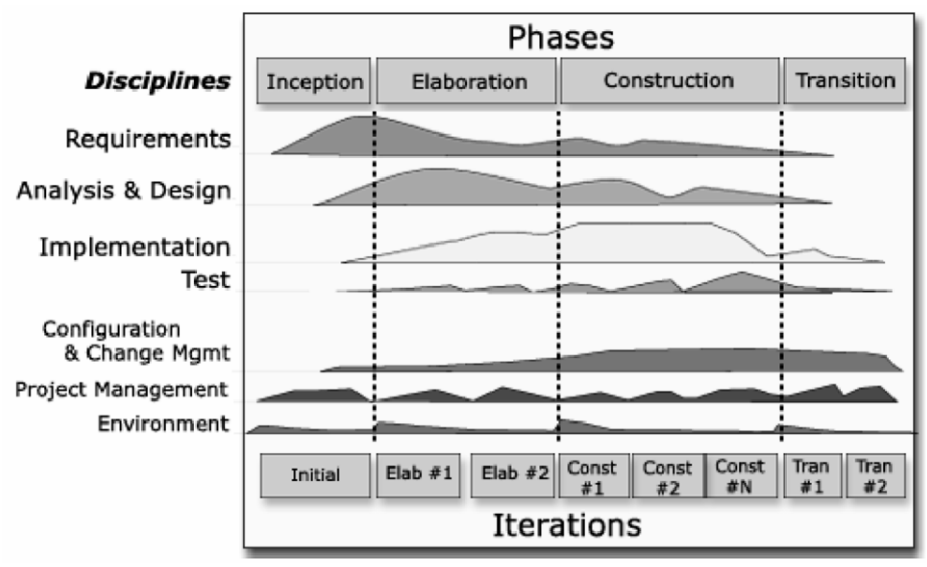


Figure 2-4: RUP Hump Chart[124]

Figure 2-4, commonly referred to as the *hump chart* diagram, depicts the typical project lifecycle attention span. The horizontal *humps* for each discipline give a rough estimate of the relative effort for each throughout the four phases. For example you can see that a large portion of Business Modelling takes place in Inception, although it does continue through to early Transition. Work on deployment usually does not start until Elaboration and doesn't really kick into high gear until the middle of Construction.[125]

Alhir[126] advises however not to standardise and enforce the Unified Process, but empower people to leverage of it. Focus on teams because when teams succeed, their projects succeed using *their* process. Likewise, the Unified Process is *scientifically* defined, but *artistically* applied.

The Enterprise Unified Process (EUP)[127] is an extension to the Rational Unified Process (RUP) that include two new phases, Production and Retirement, and several new disciplines: Operations and Support and the seven enterprise disciplines: Enterprise Business Modelling, Portfolio Management, Enterprise Architecture, Strategic Reuse, People Management, Enterprise Administration, and Software Process Improvement.

The Agile Unified Process (AUP)[128] is a simplified version of the Rational Unified Process. It describes a simple, easy to understand approach to developing business application

[124] Eeles, P. 2005. RUP for Successful J2EE Projects.

[125] Ambler, S.W. 2005. A Manager's Introduction to The Rational Unified Process (RUP).

[126] Alhir, Sinan S. 2002. Understanding the Unified Process.

[127] Ambler, S.W. 2005. A Manager's Introduction to The Rational Unified Process (RUP).

[128] Ambler, S.W. 2005. A Manager's Introduction to The Rational Unified Process (RUP).

software using agile techniques and concepts yet still remaining true to the RUP. Where the RUP is described in 3000+ HTML pages, the AUP is described in less than 30.

The EUP and AUP adaptations are steered by Scott Ambler, while the Essential Unified Process (EssUP)[129] is an adaptation by Ivar Jacobson that combines several of the Agile methodologies with CMMI into a practical reference card metaphor approach.

2.6 Revolutionary Era

Brooks[130] broke several myths, created an important awareness and also left the industry somewhat naked, without having had any remedies at hand. Many resorted to his *Mythical Man-Month Methodology*[131] defined as the set of guidelines listed below:

- *Tar Pits*. Software projects are perhaps the most intricate and complex of all man-made things. The tar pit of software engineering will continue to be sticky for some time to come.
- *Creative Slowness*. More projects have gone awry for lack of calendar time than for all other reasons combined. Programming is a creative process, like art and music. It is a creative process as opposed to a mechanical, productive process.
- *Brooks' Law*. Adding manpower to a late software project makes it later. People and time are not interchangeable. There are certain processes that can't be hurried along. Adding more people increases inter-communication and training overhead, as well as disrupts progress.
- *The Second System Effect*. The second is the most dangerous system a person ever designs; the general tendency is to over-design it.
- *The Tower of Babel*. Schedule disaster, functional misfit, and system bugs all arise because the left hand does not know what the right hand is doing. Teams drift apart in assumptions.
- *Natural Decay*. System entropy rises over a lifetime. Repairs tend to destroy structure and increase disorder.

[129] Jacobson, I. Wei Ng, P. Spence, I. 2006. The Essential Unified Process.

[130] Brooks, F.P. 1975. The Mythical Man-Month: Essays on Software Engineering. Addison-Wesley.

[131] Aron Trauring. 2003. Software Methodologies – Battle of the Gurus.

In the advent of the Information Age, as management gurus slowly came to realise the various shortcomings of their mechanical and hierarchical views of organisations, an uprising started amongst the expanding network society.[132]

In 1999 a quartet wrote *The Cluetrain Manifesto* as a revolt against the impersonal abstract trend in electronic commerce.

Online Markets...	...People of Earth
Networked markets are beginning to *self-organize* faster than the companies that have traditionally served them. Thanks to the web, markets are becoming better informed, smarter, and more demanding of qualities missing from most business organizations.	The sky is open to the stars. Clouds roll over us night and day. Oceans rise and fall. Whatever you may have heard, this is our world, our place to be. Whatever you've been told, our flags fly free. Our heart goes on forever. People of Earth, remember.

The Cluetrain Manifesto (http://www.cluetrain.com/#manifesto)

The declaration is further qualified with 95 theses. In 2001 more than a dozen software industry experts came together to make a pledge with the world whereby they acknowledge the mistakes of following hard problem solving methods as opposed to focusing more on *soft methods*[133].

We are uncovering better ways of developing software by doing it and helping others 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 there is value in the items on the right, we value the items on the left more.

The Agile Manifesto (www.agilemanifesto.com)

The Agile Alliance is spreading the human-centric approach far and wide, while adding additional depth and understanding to the human issues behind software development in

[132] Castells, M. 1999. The Rise of the Network Society.

[133] Flood. Jackson. 1991. Creative Problem Solving.

general. Highsmith[134] uses the concept of an *Agile Ecosystem* to stress the bigger picture of what software development is about. The Agile Development Methodology as it is often and commonly referred to, is actually not a methodology in itself. It is a family of methodologies that underwrites the following common set of *Agile principles*:

- The highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals.
- Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development.
- The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity, the art of maximizing the amount of work not done, is essential.
- The best architectures, requirements, and designs *emerge from self-organising teams*.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

By the time that the Agile Manifesto was signed, there already existed many supporting methodologies such as *Extreme Programming* (XP) and *Scrum*. Unlike the monolithic *heavyweight* RUP approach, they acknowledge that one size *does not* fit all, and so they instead provide a variety of *lightweight* methods that *fit* under the unifying Agile umbrella.

[134] Highsmith, J. 2002. What Is Agile Software Development? CrossTalk.

The Agile Project Management *Declaration of Interdependence* followed early in 2005.

> We …
>
> *increase return on investment* by
>
> -- making continuous flow of value our focus.
>
> *deliver reliable results* by
>
> -- engaging customers in frequent interactions and shared ownership.
>
> *expect uncertainty* and manage for it through
>
> -- iterations, anticipation and adaptation.
>
> *unleash creativity and innovation* by
>
> -- recognizing that individuals are the ultimate source of value, and
> creating an environment where they can make a difference.
>
> *boost performance* through
>
> -- group accountability for results and
> shared responsibility for team effectiveness.
>
> *improve effectiveness and reliability* through
>
> -- situationally specific strategies, processes and practices.

Declaration of Interdependence (http://pmdoi.org)

The title *Declaration of Interdependence* (DOI) has multiple meanings. It means that project team members are part of an interdependent whole and not a group of unconnected individuals. It means that project teams, their customers, and their stakeholders are also interdependent. Project teams who do not recognize this interdependence will rarely be successful.

These six values also form an interdependent set. While each is important independent of the others, the six form a system of values that provides a modern view of managing projects, particularly the complex, uncertain ones. The six statements -- value, uncertainty, customers, individuals, teams, and context -- define an inseparable whole. It is difficult to deliver value without a customer who values something. It is difficult to have viable teams without recognising individual contributions. It is difficult to manage uncertainty without applying situational specific strategies.

Each of the value statements has a distinct form -- why the *item* is important precedes the description of the *value*. So, "increasing return on investment" is why focusing on continuous flow of value is important. The value statements emphasize the importance of delivering reliable results, managing uncertainty, unleashing creativity and innovation, boosting performance, and improving effectiveness.

Each of the *means* statements conveys what this group thinks are the most important aspects of modern project management, and they also attempt to differentiate an agile-adaptive style of project management. For example, in the last value statement, the phrase "situationally specific strategies, processes, and practices," indicates that these items should not be overly standardized and static, but dynamic to fit the needs of projects and teams. Other styles of project management are more prone to standardization and prescriptive processes.

In order to consistently deliver successful results, great project leaders should embrace the following core principles:

- *Relentlessly Focus on Value*. Focus efforts on generating organisational value rather than managing tasks.
- *Be Situational Specific*. Use situationally specific strategies, not a one-size-fits-all approach.
- *Manage Uncertainty*. Manage uncertainty through client focused collaborative exploration and proaction.
- *Continuously Align to Changing Situations*. Choose strategies for leading within a dynamic environment.
- *Lead with Courage*. Confront reality with conviction and a dedication to purpose.
- *Build Strategies that Leverage People*. Challenge team members with opportunities to grow professionally.
- *Design Strategies Based on Teamwork*. Develop and sustain a collaborative team environment.
- *Communication Through Immediate and Direct Feedback*. Maintain control through feedback, not prescriptive plans.

None of these principles are new to software development management literature.[135] These value statements and principles seem to answer most of the concerns to date, but can it be done? How do successful software companies do it today? Google's leaders say that they uphold innovation through small, highly motivated groups of bright people who are given access to immense resources.[136]

Anderson[137] claims that it is possible to be an agile project manager and be running a CMMI compliant process. He found that the DOI is fully compatible with CMMI and there is nothing in any of the 25 process areas in the CMMI which is incompatible with the DOI. The problems seem to arise at the interpretation of the specific process guidelines. The DOI was borne out of observation that general project management practices were leading to undesirable behaviour and unfavourable results. The DOI seeks to *reset the mindset or framework* of how people think about projects so that they adopt the correct behaviour that leads to good results.

Many modern development methodologies that sprouted over that last decade complement various aspects of the Agile Manifesto. A partial list of methods that support the Agile Manifesto is:

- Extreme Programming (XP)
- Dynamic Systems Development Method (DSDM)
- Scrum
- Feature-Driven Development (FDD)
- Crystal Clear
- Adaptive Software Development (ASD)
- Lean Software Development
- Agile Unified Process (AUP)

The various agile methods are focused on different aspects of the software development life cycle. For example, Extreme Programming is more focused on practices, while Scrum focuses on managing the software projects. DSDM and AUP provide full coverage over the development life cycle.

[135] Davis, Alan M. 1995. 201 Princples of Software Development.
[136] Vise, David. 2005. The Google Story:287.
[137] Anderson, D.J. 2006. CMMI DOI Comparison.

For the purpose of the thesis, the focus will however be on XP and Scrum as these complement one another well.[138]

2.7 Extreme Programming (XP)

When Extreme Programming was made public knowledge it caused excitement and resistance. XP is not a sequential process methodology. Since the early 1990s Kent Beck was working together with Ward Cunningham,[139] who devised Class-Responsibility-Collaboration Cards (CRC Cards). Using CRC Cards is a simple yet effective software design technique that is still widely practiced today. At that time Beck and Cunningham gained experienced in an approach to software development that made everything seem simple and more efficient. Beck spearheaded a project at DaimlerChrysler in 1996, where they applied their radically new concepts in software development.

Originally they identified four values and the fifth was added more recently: Communication; Simplicity; Feedback; Courage; and Respect. These are the values sought out by XP team members.

Extreme Programming demands a high discipline and commitment, despite its emphasis on flexibility and human-centeredness. It brings the whole team together with simple practices at a sustainable pace, with enough feedback to enable the team to learn effectively and act efficiently. The general attributes of XP are:

- *Small whole teams*. Only 3 to 10 active team members.
- *Coached*. A team led by a coach.
- *Onsite domain experts*. One or several customers providing ongoing expertise.
- *Minimal documentation*. The unit of specification is a *user story*.
- *Short iterations*. Two, three or four-week iterations, with a demonstrated working integrated system, and a working system build delivered to customers at the end of every two to five iterations.

The details are defined by the twelve XP practices that follow.[140]

[138] Mar, K. Schwaber, K. 2002. Scrum with XP.

[139] Beck, K. 1999. Extreme Programming Explained: Embrace Change. Addison-Wesley.

[140] Beck, K. Fowler, M. 2002. Planning Extreme Programming.

2.7.1 Planning game

The planning process shown in Figure 2-5 allows the customer to define the business value of desired features, and uses cost estimates provided by the programmers, to choose what needs to be done and what needs to be deferred. The effect of the planning process is that it is *easy to steer* the project to success. Beck uses driving a car as a metaphor for explaining the planning game. Software development is a process. It can go well, or it can go badly. To keep it going well *managers must continually direct it*. To direct it they must frequently assess the direction it is going, compare this to the direction they want it to go, and then make careful adjustments.[141]

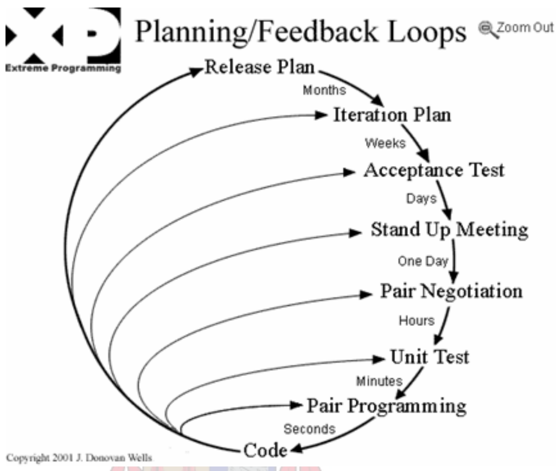


Figure 2-5: XP Planning and Feedback diagram[142]

2.7.2 Small releases

The team puts a simple system into production early, and update it frequently on a very short cycle as shown in Figure 2-6.

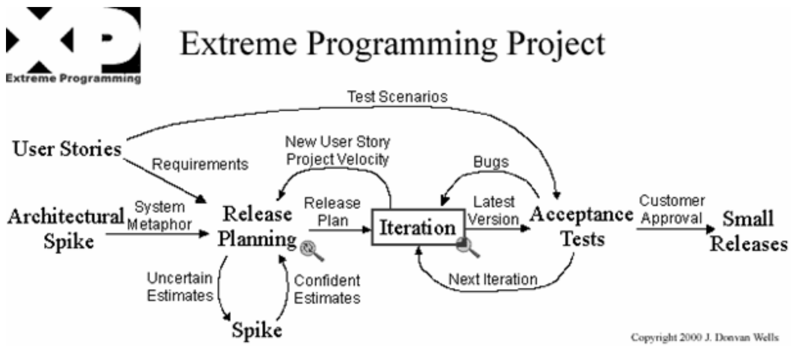


Figure 2-6: XP Project diagram[143]

[141] This behaviour is well defined and described in the field of Cybernetics as expanded in paragraph 3.6 on page 90.

[142] Wells, J.Donovan. Extreme Programming Website http://www.extremeprogramming.org

[143] Wells, J.Donovan. Extreme Programming Website http://www.extremeprogramming.org

2.7.3 Metaphor

Teams use a common *system of names* and a common system description that guides development and communication. It provides project vision.

2.7.4 Simple design

Seek the simplest solution that meets the current requirements. The focus is on providing *business value now*, opposed to future proofing.

2.7.5 Testing

Teams focus on continuous validation of the software using test driven approach to designing systems, by writing tests first, then solutions that meets the requirements reflected in the tests. In addition, *customers provide acceptance tests* that enable them to be certain that the features they need are provided.

2.7.6 Refactoring

Teams continuously improve the design and implementation of the system by keeping the code neat and optimised without compromising completeness.

2.7.7 Pair programming

Programmers program in pairs, working together at one machine. Pair programming has been shown by many experiments to produce better software at similar or lower cost than programmers working alone.

2.7.8 Collective ownership

Whole team responsibility resolves many delays and conflicts as shown in Figure 2-7.

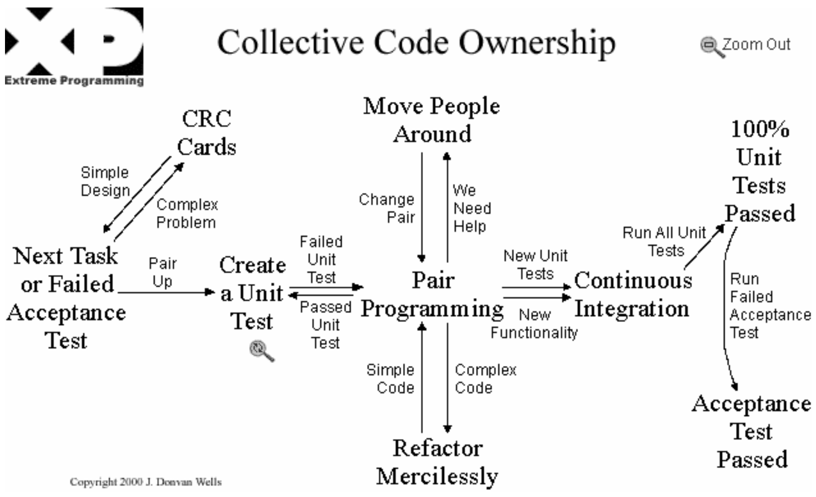


Figure 2-7: XP Collective Code Ownership[144]

[144] Wells, J.Donovan. Extreme Programming Website http://www.extremeprogramming.org

2.7.9 Continuous integration

Teams integrate and build the software system as often as possible. This keeps everyone's work synchronised. Integrating more frequently eliminates integration problems that plague teams who integrate less often.

2.7.10 40-hour weeks

People make more mistakes when they are overworked.

2.7.11 Onsite customer representative

The project is steered by a dedicated individual who is empowered to determine requirements, set priorities, and answer questions as needed. The effect of being onsite is that communication improves, with less hard-copy documentation - often one of the most expensive parts of a software project.

2.7.12 Coding standards

Consistency is essential for a team to work effectively in pairs, and to share ownership.

The quality of the software produced is considered to be the highest priority of an XP development team. However the compromise for constantly achieving high quality is scope. Table 2-1 illustrates the variance of the four key project variables.[145]

Variable	Traditional Projects	XP Projects
Scope	Fixed	Variable
Time	Fixed	Fixed
Cost	Fixed	Fixed
Quality	Variable	Fixed

Table 2-1: Project variables

The Theory of Constraints (TOC)[146] and Six Sigma (6σ)[147] methods could be applied to ensure these variances are controlled. However these methods rely on repeatable and predicable processes such as rated at CMMI level 5. XP achieves it by applying its five core values and based on the assumption that quality, fixed-cost and on-time delivery is more critical than scope. Software is often bloated with unnecessary features and by adding code

[145] Rooney, D. Martin, N. 2003. Enabling Software Quality with Extreme Programming.

[146] Theory of Constraints is a thinking process for the effective management of organisations and systems developed by Eliyahu M. Goldratt.

[147] Six Sigma is a quality management control technique used to measure process and output deviations in order to drive it towards expected normal deviation. George, M. Rowlands, D. Kastle, B. 2004. What is Lean Six Sigma? McGraw-Hill.

one puts the overall product quality at risk. The XP approach therefore seems the natural choice.

The quality of the code is maximized through test driven development and unit testing. The quality of the overall system is maximized through continuous integration. The quality of the system from the business perspective is maximized through iterative development and frequent small releases, which allows for functional testing at much smaller intervals. All of these aspects of XP provide feedback in very short cycles to the development team and business clients, providing a clear picture of the progress of the system's development and allowing for adaptation to changes in the business environment.[148]

Theunissen[149] confirmed via a case-study and survey that Equinox Financial Solutions had successfully adopted XP and investigated the possibilities for Telkom SA to turn from its Rational Unified Process adoption towards a more agile methodology.

2.8 Scrum

The root of the Scrum and Rugby metaphors goes back to 1982 when Takeuchi and Nonaka's wrote an article for the 75th Anniversary Colloquium of the Harvard Business School on the unique features of the new product development process within Japanese companies.[150] In their later article in the Harvard Business Review of 1986, *The New New Product Development Game*, they argued that the rugby-style approach has tremendous merit in terms of *speed and flexibility*[151] – in a single word *agility*. The article became the cornerstone of the agile approach to software development developed by Ken Schwaber and Jeff Sutherland developed during the mid 1990s.[152]

The Rugby ball being passed around represents a shared understanding of the vision and mission. It embraces highly subjective insights, intuitions and hunches. Unlike a baton being passed in a predetermined sequence from one runner to the next in a relay race, if one had to trace the movement of the Rugby ball throughout the game it would seem to move in an unpredictable chaotic manner, leaning towards the theories of Brownian motion, Artificial Life and Complex Adaptive Systems. The ball movement is determined on the spot, here and now, by the team members' inner-game experiences, micro-tactics and skills. For this reason

[148] Rooney, D. Martin, N. 2003. Enabling Software Quality with Extreme Programming.

[149] Theunissen, W.H.M. 2003. A case-study based assessment of Agile software development.

[150] Nonaka, I. Takeuchi, H. 1995. The Knowledge-Creating Company.

[151] Nonaka, I. Takeuchi, H. 1995. The Knowledge-Creating Company:93.

[152] Schwaber, K. 2001. Agile Software Development with SCRUM. Prentice-Hall

Nonaka and Takeuchi also warns that this approach has an over-reliance on the complexity of socialization which becomes inefficient as the number of team players increase.[153]

Scrum offers an empirical approach, which allows team members to work independently and cohesively in a creative environment. It recognises the importance of the social aspect in software engineering: the name derives from the game of rugby, and refers to a rugby play in which the forwards of each side come together in a tight formation and struggle to gain possession of the ball when it is tossed in among them. The process is quick, adaptive, and self-organizing, and it represents a significant change from sequential development processes. The Scrum proponents believe that software should not be developed according to well defined repeatable processes used in typical manufacturing. This repetition makes the input and output parameters more predictive and descriptive, but this is not a helpful goal in today's software engineering. Time to market, return on investment, and the need to build a vision alongside the customer are among the major challenges in modern software engineering.[154]

Scrum is an Agile Software Development process designed to add energy, focus, clarity, and transparency to project teams developing complex systems. It leverages Artificial Life research by allowing teams to operate close to the edge of chaos to foster rapid system evolution by enforcing a simple set of rules that allows rapid self-organisation of software teams to produce systems with evolving architectures. According to Sutherland a properly tuned Scrum team will:[155]

- increase speed of development;
- align individual and organisation objectives;
- create a culture driven by performance;
- support shareholder value creation;
- achieve stable and consistent communication of performance at all levels; and
- enhance individual development experience and quality of life.

Scrum uses similar practices as Extreme Programming, including short daily stand-up meetings, time-boxed iterations called sprints, and the 40-hour week. While Extreme

[153] Nonaka, I. Takeuchi, H. 1995. The Knowledge-Creating Company:118.

[154] Krebs, Joe. 2005. RUP in the dialogue with Scrum.

[155] Sutherland, J. Viktorov, A. Blount, J. 2006. Adaptive Engineering of Large Software Projects with Distributed/Outsourced Teams.

Programming is very specific in how the work should be conducted with pair-programming and continuous integration, Scrum leaves it up to each team to decide the best way to accomplish the required work. It should therefore be combined with more process prescriptive methodologies such as XP to provide a more complete and defined approach to systems development.[156]

The primary philosophy of Scrum is to manage *detailed complexity*[157] by dividing the project into smaller and more manageable parts. This strategy is similar to the practice of *Getting-Things-Done*[158] by creating prioritised lists and then moving swiftly, quickly adjusting focus and tackling each item one-by-one.

Scrum is an iterative methodology in which each iteration is called a Sprint. The prioritised list of product requirements is called a Product Backlog. Items is carefully selected from the Backlog and allocated to a Sprint. The tasks in the Sprint are then to be completely finished at the Sprint Review Meeting at the end of the Sprint when the result of the work is presented to the customers. For the code to be considered finished, it has to be written, implemented, thoroughly tested and ready for shipping to the customer.

2.8.1 Scrum roles

There are only three different roles in a Scrum project, the Product Owner, the Scrum Master and the Team. These three roles are described below:

- *Product Owner* – this is the customer representative responsible for listing the requirements in the Product Backlog and to prioritize them. With the help of the Team he chooses the requirements that will be implemented during the Sprints and during the sprint it is a benefit if he is available for further questions from the Team. He is not however allowed to contact the Team and ask them to sneak in another task in the sprint unless he is willing to remove another.
- *Scrum Master* – the coach that guides the Team in the right direction so that it concentrates at the tasks committed to the Sprint and follows the guidelines of Scrum. He is responsible for protecting the Team from outside disturbances during the sprint such as other commitments both in the other projects as well as in the Scrum project.

[156] Schwaber, K. Mar. K. 2002. Scrum with XP.

[157] Senge, P. 1994. The Fifth Discipline – The Art & Practice of the Learning Organization.

[158] Allen, David. 2002. Getting Things Done – The art of stress-free productivity.

The Scrum Master is also in charge of conducting Daily Scrum meetings with the Team.

- *The Team* – is the developers that work on the project. They are self-managed and independent. They help the Product Owner to sort out the requirements for the sprints and then they break down the requirements into manageable tasks that are introduced into the Sprint Backlog. They meet every day at Daily Scrum meetings and inform the others of what they have done, what they are going to do next and possible problems they have encountered so far. A well organised team should not consist of more than seven developers. If the project is larger than this it is better to divide the developers into several teams and have them work on different functionality.

2.8.2 Scrum artifacts

There are only three small work products defined in Scrum, the Product Backlog belongs to the Product Owner, the Sprint Backlog to the Team and the Scrum Master is responsible for the Burn-down Chart. In addition to this every project is welcome to use whatever documentation they want.

- *Product Backlog* - this is the document that contains all of the requirements. It belongs to the Product Owner and he is responsible for keeping the Product Backlog up to date during the project; changing or deleting requirements that are out of date and adding new requirements when such are found. The Product Owner is the only one who has a right to change anything in the Product Backlog but it is the Team that estimates the work-time needed on each requirement. In the Product Backlog the requirements should not be very detailed and the estimations for the requirements are in workdays.

- *Sprint Backlog* – when the Product Owner and the Team have decided what is going to be included in the sprint the Team breaks down the chosen requirements into smaller tasks and estimates the time needed for each one. These tasks should be small, no more than 4-16 man-hours, in order to be manageable for the developers assigned to it. These tasks are recorded in a document called at Sprint Backlog and during the sprint it is the Team's responsibility to keep it up to date with the remaining time on the task and who is assigned to it. The Team should also insert new tasks that are found during the sprint that are required to be done before another task in the sprint.

- *Burn-down Chart* – this daily graph is a visual aid to show how much work remains in the Sprint or in the project. It is a graph with the estimated time of the remaining work versus the time left in the sprint. It is a way for all of the stakeholders to see how the work is progressing and it implies if the Team is going to be able to keep their goal or if they have to contact the Product Owner and redo the prioritizing. It can be compared to a visual graph in Extreme Programming over estimated time remaining versus time left in the sprint.

2.8.3 Scrum meetings

Scrum advocates the principle that different projects need different approaches for the work. It is up to the Team if they want to practice pair programming or how they want to test their products. Scrum does however advocate the 40-hour week explained under the practices of Extreme Programming and also the four meetings explained below, to be able to organize the work. All meetings in Scrum are time-boxed so as to not stray from the subject.

- *Sprint Planning Meeting* – this meeting consists of two parts and both parts are time-boxed to four hours each. During the first part the Product Owner explains all of the entries in the Product Backlog, what they mean and why they are necessary. After that the Team gives a rough estimate of how much time each entry will take in days. Then the Product Owner will have to choose what is most important to produce during the sprint. Here the Product Owner and the Team have to compromise some, the Product Owner thinks about what is important from a sales view and the Team looks at what has to be done first from a technical view. During the second half of the meeting the Team breaks down the requirements in smaller tasks, about 4-16 ideal man-hours, and estimates the work time on each task. All of the tasks are entered into the Sprint Backlog and the Team members decide what they want to start to work on. The Scrum Master often works as a documenter while the Team discusses the tasks and time-estimation.

- *Sprint Review Meeting* – during this meeting the Team presents the work they have been working on during the last sprint. Everything that is presented has to be finished, i.e. it should be ready to be used by the customer. They should not just finish coding but they should also have finished testing the product and have verified that it works correctly. This meeting is also time-boxed to four hours so as not to get into discussions that do not have anything to do with the work during the sprint.

- *Sprint Retrospective Meeting* - during this very important meeting the Team discusses the work during the last sprint. What worked well, what did not, what should have been done differently and what can be done not to make the same mistakes again? It gives the Team a chance to decide how they can do to make the next sprint work better and to not make the same mistake twice. The Sprint Retrospective Meeting is, just as all the other meetings, time-boxed as not to stray from the issue at hand. In this case the meeting should not be more than three hours long.
- *Daily Scrum Meeting* – every day the Scrum Master and the Team have a 15 minute long stand-up meeting. All of the team members answer these three questions:
    - What was done since the last Daily Scrum?
    - What is planned to get done between now and the next Daily Scrum?
    - What is in the way that is preventing work from getting done?

  The purpose is that the Team should discuss the answers with each other and not report to the Scrum Master. The Scrum Master is responsible for calling the meeting and for making sure that everyone participates but The Daily Scrum Meeting is for the Team. In order to keep the meeting to 15 minutes you do not discuss solutions in the Daily Scrum. If someone is having a problem that someone else can help with you decide to meet after the meeting and discuss it then.

Several organisations have reported success with Scrum. Teleca, Sweden, has started to use Scrum in two of their projects and the intermediate results are positive.[159] Sun Space & Information Systems, South Africa, has helped his company and other to adopt Scrum successfully.[160] Schwaber[161] states that it is straight forward and simple to implement Scrum. When implementing Scrum management with some Agile processes, satisfactory releases is eminent, everyone will enjoy working with each other, overtime will be eliminated, customers will be pleased, the competition will be vanquished, and bonuses will flow. Schwaber however also warns that it is still No Silver Bullet.

[159] Linder, G. 2006. Evaluation of Software Development Projects using the Agile Method Scrum.

[160] Confirmed via formal discussions and e-mail correspondence with Jaco van der Merwe, Head of Information and Software Systems division, Sun Space & Information Systems, Stellenbosch South Africa.

[161] Schwaber, K. Martin, R. C. Best Practices in Scrum Project Management and XP Agile Software Development.

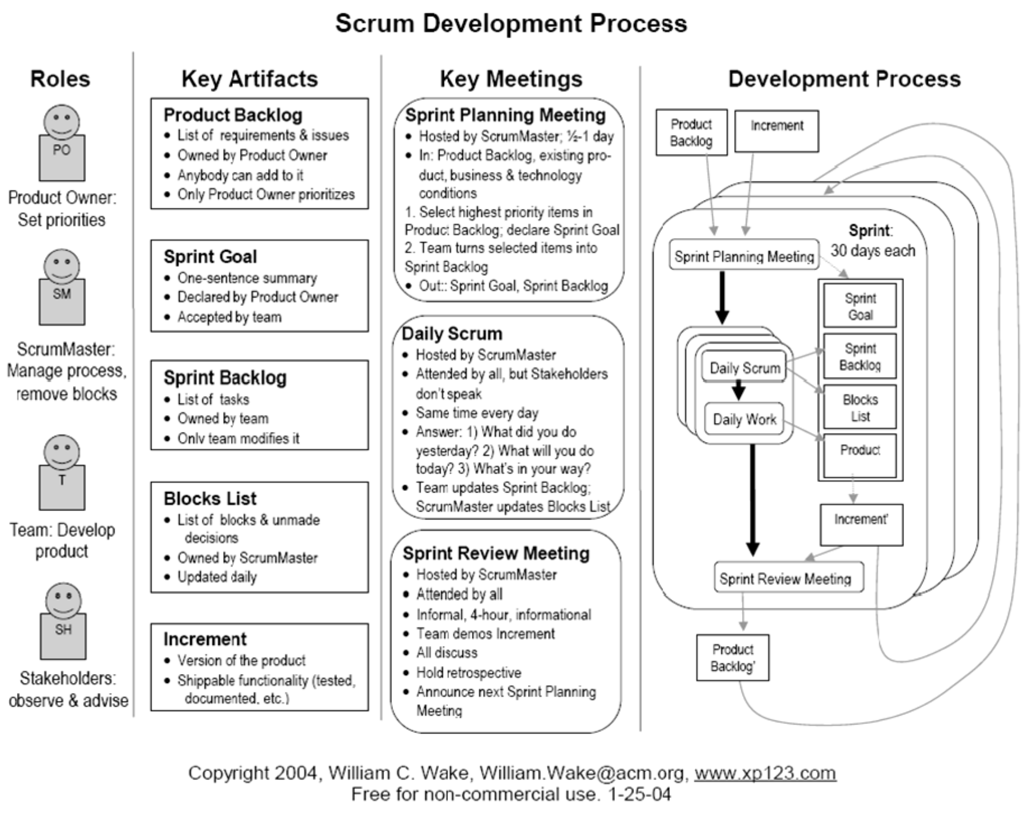

Scrum Development Process
Roles
PO
Product Owner: Set priorities
SM
ScrumMaster: Manage process, remove blocks
T
Team: Develop product
SH
Stakeholders: observe & advise
Key Artifacts
Product Backlog
• List of requirements & issues
• Owned by Product Owner
• Anybody can add to it
• Only Product Owner prioritizes
Sprint Goal
• One-sentence summary
• Declared by Product Owner
• Accepted by team
Sprint Backlog
• List of tasks
• Owned by team
• Only team modifies it
Blocks List
• List of blocks & unmade decisions
• Owned by ScrumMaster
• Updated daily
Increment
• Version of the product
• Shippable functionality (tested, documented, etc.)
Key Meetings
Sprint Planning Meeting
• Hosted by ScrumMaster; ½-1 day
• In: Product Backlog, existing product, business & technology conditions
1. Select highest priority items in Product Backlog; declare Sprint Goal
2. Team turns selected items into Sprint Backlog
• Out:: Sprint Goal, Sprint Backlog
Daily Scrum
• Hosted by ScrumMaster
• Attended by all, but Stakeholders don't speak
• Same time every day
• Answer: 1) What did you do yesterday? 2) What will you do today? 3) What's in your way?
• Team updates Sprint Backlog; ScrumMaster updates Blocks List
Sprint Review Meeting
• Hosted by ScrumMaster
• Attended by all
• Informal, 4-hour, informational
• Team demos Increment
• All discuss
• Hold retrospective
• Announce next Sprint Planning Meeting
Development Process
Product Backlog
Increment
Sprint: 30 days each
Sprint Planning Meeting
Sprint Goal
Sprint Backlog
Blocks List
Product
Daily Scrum
Daily Work
Increment'
Sprint Review Meeting
Product Backlog'
Copyright 2004, William C. Wake, William.Wake@acm.org, www.xp123.com
Free for non-commercial use. 1-25-04

Figure 2-8: Scrum Development Process

The Scrum process flow that is summarised in Figure 2-8 is a process of change. Change within the engineering organisation, in the tools employed, practices followed, sociology of interaction, workspace, and ongoing collaboration in creating increments. Change between the engineering organisation and marketing organisation in how work is planned and changes are handled. Change in the way progress is perceived, as it is tracked by requirements rather than tasks.[162]

Looking at release plans more as forecasts allows people to consider adjustments due to various circumstances and make better decisions about moving forward. It is more a roadmap for knowing where the team is and looking ahead to what is coming next. The hardest change is backsliding. Letting others figure out how to work together rather than telling them how to work together is a major shift in management philosophy. This shift requires the Scrum Master to continuously observe and coach his staff to adopt the self-management approach. Every team member does the best that can be done. And by understanding and seeing that the product at the end of each Sprint is only as good as everyone's cooperative effort is also a humbling experience in a profession where people try to be heroes. All of

[162] Schwaber, K. Martin, R. C. Best Practices in Scrum Project Management and XP Agile Software Development.

these changes are gradual. As soon as the changes appear to have taken place, some stress occurs and everyone reverts back to form. Regaining the progress each time causes strain.[163]

Schwaber and Schatz realized that the problems would never end and that the core of Agile processes is unearthing problems so that they can be solved; and in a changing, complex development environment, these problems would only end when changes ended, and that would be never. Making the decision to adopt Agile is a commitment to change the team culture. Someone has to be willing to be the champion and take risks in order to improve the organisation. It must start with a vision, a realization of the pain, and a willingness to question everything in your development process. [164]

Toyota's success has been attributed to fourteen clearly identified principles. Boris Gloger[165] shows that Scrum is fully compatible with these principles as outlined in The Toyota Way[166]. Toyota rejected ISO 9001 and reverted to their preferred way defined by their principles as follows:

- Decisions are based on a long-term philosophy even at the expense of short-term financial goals.
- Continuous process flow brings problems to the surface.
- A pull or demand driven system avoids overproduction.
- Workload should be levelled out.
- The culture must foster a drive for fixing problems to *get quality right the first time* and minimise work stoppages.
- The foundation for continuous improvement and employee empowerment is driven by standardising tasks.
- Problems are made transparent through the use of visual controls.
- Reliable technology that serves the teams objectives and processes will be adopted.
- Leaders should thoroughly understand the work, live the philosophy, and teach it to others.

[163] Schwaber, K. Martin, R. C. Best Practices in Scrum Project Management and XP Agile Software Development.

[164] Schwaber, K. Martin, R. C. Best Practices in Scrum Project Management and XP Agile Software Development.

[165] Gloger, Boris. 2006. Comparison of Methodologies.

[166] Liker, Jeffrey K. *The Toyota Way*.

- Exceptional people and teams will be developed to follow the company's philosophy.
- Thorough understanding of every situation.
- Decisions are carefully taken by consensus, with thorough consideration of all options, and implemented rapidly.
- A learning organisation is build through relentless reflection and continuous improvement.

Scwaber predicts that only about one-third of companies that try to use Scrum will succeed. Those companies that do succeed with it are often those for whom technology is their lifeblood, and if they don't succeed with it, they risk going out of business.[167]

Managing even a single Scrum team can be difficult at times; managing multiple Scrum teams presents an even greater challenge. Siemens Communications believes they found a way to make it easier. They established several Scrum teams to work collaboratively in creating one software system and then soon recognised that this collection of Scrum teams formed its own Agile Development System. Because the Theory of Constraints (TOC) works so well on optimising value-creating systems, they wondered if it would have a similar effect on the Agile development system and found it extremely powerful and valuable in increasing their system's velocity.[168]

2.9 Socialistic Era

Unlike most other Agile methods, free software and open-source development does not have methodology gurus (they have gurus of other sorts). What emerged was community-owned open-source software products of outstanding quality. These communities are strung together by a passion for developing solutions for which the requirements never existed. Eric S. Raymond's famous essay, and later his book, titled *The Cathedral and the Bazaar*,[169] explained that open-source development was so successful by being human-centric, iterative, incremental working systems and build with pride. It is not resource-limited. There is no budget and no deadlines. Nonetheless, the enormous success of projects like the GNU toolset, Linux, Apache, Mozilla Firefox, and so on confirms the open-source phenomena's value and robustness in large complex and mission critical software projects. Very valuable and

[167] Baxter, Andrew. 2005. Rapid results without a rugby scrum.

[168] Pichler, Roman. 2006. Agile Gets Lean – How we optimized our Agile Development System using the Theory of Constraints and Scrum.

[169] Raymond, E. S. 2001. Cathedral and the Bazaar – Musings on Linux and Open Source by an Accidental Revolutionary.

successful companies such as Thawte[170], Google and Amazon build their businesses on these foundations. Today even financial institutions are turning towards open-source solutions. A host of very active and valuable open-source ERP and CRM projects such as Compeire[171], Sequoia[172], and the MIT Open for Business[173] project are competing with large commercial packages such as SAP R/3, Oracle, Sage and Microsoft Dynamics. Telecoms platforms such as Asterisk[174] are powering more and more businesses. Open Source Software is now being adopted by corporations much faster than a decade ago.

The following is a selection of CatB[175] lessons that map closely with Agile principles:

- *Commitment.* Every good work of software starts by scratching a developer's personal itch. If you have the right attitude, interesting problems will find you. When you lose interest in a program, your last duty to it is to hand it off to a competent successor.
- *Refactoring.* Good programmers know what to write. Great ones know what to rewrite and reuse. Perfection in design is achieved not when there is nothing more to add, but rather when there is nothing more to take away. Plan to throw some away; you will, anyhow.[176]
- *Customer on-site.* Treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging. The next best thing to having good ideas is recognising good ideas from your users. Sometimes the latter is better. Often, the most striking and innovative solutions come from realising that your concept of the problem was wrong.
- *Small releases.* Release early. Release often. And listen to your customers.
- *Continuous Integration and Testing.* Given a large enough beta-tester and co-developer base, almost every problem will be characterised quickly and the fix

[170] Mark Shuttleworth created a huge global success from a small team in Cape Town using open source software. Today his UK based company called Canonical is fuelling several open-source initiatives and boldly taking on giants such as Microsoft.

[171] See www.compeire.org

[172] See www.sequoiaerp.org

[173] See www.ofbiz.org

[174] See www.asterisk.org

[175] CatB is short for The Cathedral and the Bazaar. The lessons are taken from the Raymond's book. See www.catb.org

[176] Taken directly from Fred Brooks' *The Mythical Man-Month.*

> obvious to someone. If you treat your beta-testers as if they're your most valuable resource, they will respond by becoming your most valuable resource.

Provided the development coordinator has a communications medium at least as good as the Internet, and knows how to lead without coercion, many heads are inevitably better than one. Raymond believes that the future of open-source software will increasingly belong to people who know how to play Linus's game[177], people who leave behind the cathedral and embrace the bazaar.

The social character and philosophy of the open-source community is possibly best described by the African concept of Ubuntu symbolising *humanity towards others*.[178] In order for members of the Ubuntu[179] community to work together effectively, they laid down some *ground rules* for cooperation and behaviour to considerate, respectful, collaborative, open-minded, inquisitive, and graceful.

In contrast with the information technology giants such as Microsoft, IBM and Oracle, the open-source organisations base their predominant resource on free-spirited social networks of passionate people. The social contracts do not differ and people come and go like bees in a hive. The primary difference between the old Goliaths and the young Davids is adopting a closed-system versus open-system approach. This cultural shift has already occurred to IBM, while Microsoft is desperately trying to catch on to the socialistic era dominated by the forces of *contingent labour*[180]. It is a much different economic playground to compete in, where the rules are open and the actors are uniquely competitive.

Canonical and Google's successes sprouted around communities of developers as users and matured into mass market acceptance. It is an enormous achievement made possible by a passion for the impossible. In a turbulent sea of abundant rapid changes it is tough to invent in silos and survive. Instead, it is essential to nurture a community that sprout new cultures and innovations.

---

[177] Raymond gives all credit for CatB to Linus Torvalds the creator of Linux.

[178] Archbishop Desmond Tutu. 2000. No Future Without Forgiveness. Tutu describes Ubuntu as follows: A person with ubuntu is open and available to others, affirming of others, does not feel threatened that others are able and good, for he or she has a proper self-assurance that comes from knowing that he or she belongs in a greater whole.

[179] Ubuntu is a version of Linux being developed and marketing by Canonical, a young UK based company founded by a South African, Mark Shuttleworth. See www.abuntu.com and www.canonical.com

[180] Barley, S.R. Kunda, G. 2004. Gurus, Hired Guns, and Warm Bodies – Itenerant Experts in a Knowledge Economy. Contingent labour is a term economists and sociologists use for an array of short-term arrangements including part-time work, temporary employment, self-employment, contracting, outsourcing, and home-based work.

2.10 Metamodels

A process metamodel is a model that describes process models. It must therefore have the flexibility to describe all common development methods ranging from the traditional linear models to large and complex development methodologies. The metamodel defines a reference system from models in a consistent standardised way in terms of common concepts and associated terminology. Thus, the key to a good process metamodel is the ability to provide an optimal compromise between flexibility and standardisation.

The primary benefit of metamodels is its adaptability to changes in real-world process implementations. The cultures and structures within and amongst organisations are different. Any two companies that may have adopted the same specific standard work practices and reached comparative maturity levels would typically have tailored their processes within the context of their cultural, geographical and economical position. Furthermore as the external and internal situation changes it needs to occasionally alter its processes. These alterations could typically be done as continuous improvement programmes or radical restructuring.

Commercial tools that have to support the general landscape of SDLC methodologies need to do so as a customisable process framework. There are many such tools around today and many more that has come and gone. These tools offer development teams process guidance and integrated workflow management.

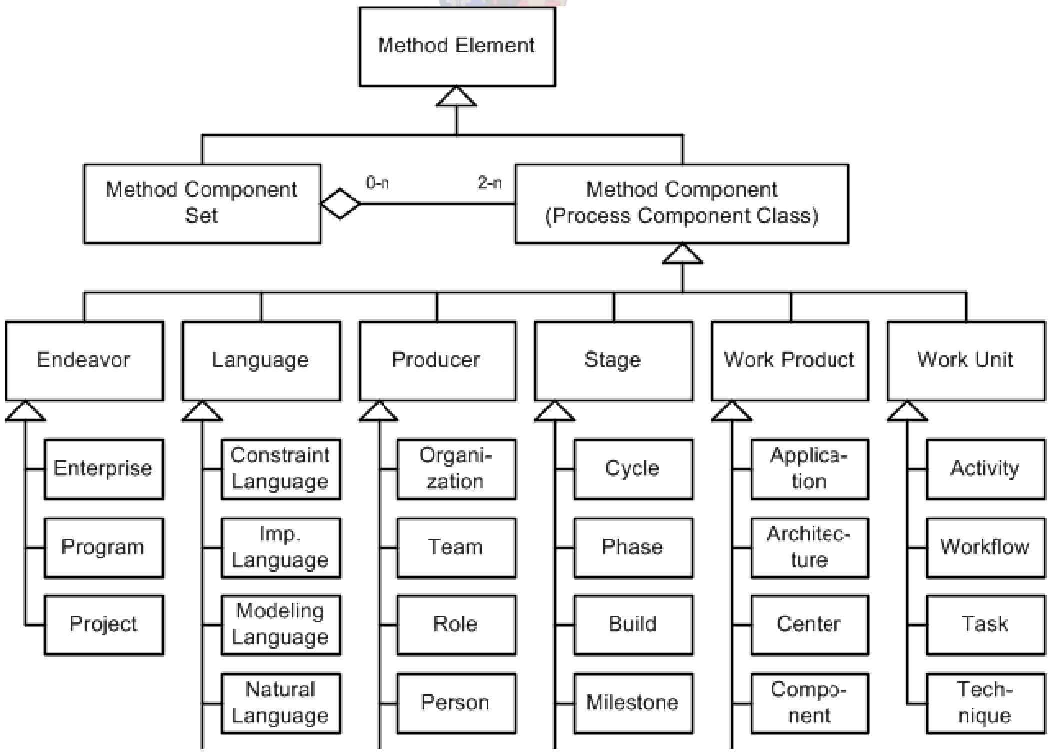


Figure 2-9: OPEN Process Framework meta-model (partial)

The more elaborate model for SDLC design shown in Figure 2-9 is part of the Henderson-Sellers OPEN Process Framework Metamodel[181] which is a free online repository. The purpose of the metamodel is to define the abstract elements and associations that are common to all methodologies. These are for example elements such as Team, Role, Milestone, Component, Task, and Project. Associations are for example that a Project is associated with many Milestones and several Tasks is allocated to each Milestone. The metamodel would also be decoupled from any specific dialect or language. Hence for some the concept of Milestone would be named Iteration, and for others it is called Sprint.

Microsoft evolved and embedded its so-called Microsoft Solutions Framework (MSF) into a viable commercial offering called Visual Studio Team System (VSTS). MSF provides the infrastructure for maintaining a set of software engineering processes, principles, and practices that empower developers to achieve more success during each step of the software development life cycle. As a metamodel, MSF provides a baseline for adaptable guidance. It is based upon experiences and best practices from inside and outside of Microsoft, to increase the chance of successful delivery of information technology solutions to the customer by working fast, decreasing the number of people on the project team, averting risk, while enabling high quality results. The MSF philosophy therefore holds that there is no single structure or process that optimally applies to the requirements and environments for all sorts of projects. Therefore MSF supports multiple process approaches, so that it can be adapted to support any project, regardless of size or complexity. This flexibility means that it can support a wide degree of variation in the implementation of software engineering processes while retaining a set of core principles and mindsets.[182]

[181] Firesmith, D. 2006. OPEN Process Framework Metamodel.

[182] Sridharan, P. 2004. Visual Studio 2005 Team System: Microsoft Solutions Framework.

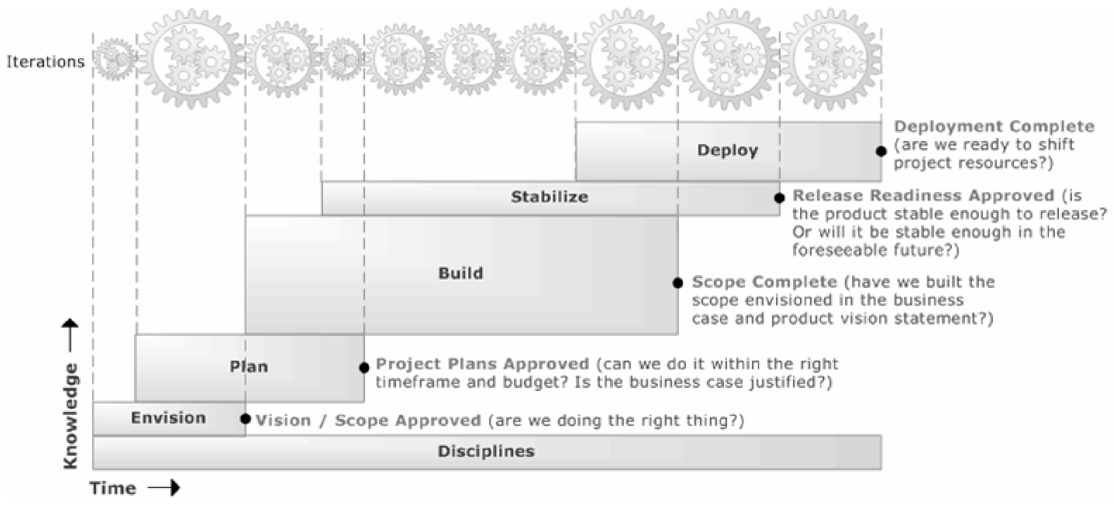

Iterations
Deploy
Deployment Complete (are we ready to shift project resources?)
Stabilize
Release Readiness Approved (is the product stable enough to release? Or will it be stable enough in the foreseeable future?)
Build
Scope Complete (have we built the scope envisioned in the business case and product vision statement?)
Plan
Project Plans Approved (can we do it within the right timeframe and budget? Is the business case justified?)
Envision
Vision / Scope Approved (are we doing the right thing?)
Disciplines
Knowledge
Time

Figure 2-10: MSF Life Cycle Model[183]

Figure 2-10 shows that the Microsoft Solutions Framework Process Model consists of series of short development cycles and iterations. The *Knowledge dimension* shows that the lifecycle practices are grounded in a set of disciplines and the various overlapping lifecycle phases correlates well with RUP. Along the *Time dimension* the model embraces rapid iterative development with continuous discovery and adaptation, driven by regular interfacing with the various project stakeholders. The overlapping phases and incremental iterations ensure a healthy, tangible flow of value defined a focused, stable portion of the overall envisioned solution.

Microsoft originally supported very limited variation of its internal SDLC practices with supporting documentation for its process compliance with ISO 9001. With the redevelopment of their development tool chain they shifted focus in support of the two dominant industry variations. MSF provides a high-level framework of guidance and principles which can be mapped to a variety of prescriptive process templates. It is structured in both descriptive and prescriptive methodologies. The descriptive component is the MSF metamodel, which is a theoretical description of the SDLC best practices for creating SDLC methodologies. Microsoft provides two prescriptive methodology templates that provide specific process guidance, named MSF for Agile Software Development and MSF for Capability Maturity Model Integration Process Improvement. [184] A third template for Scrum was developed by another ISV. These three variants will be briefly elaborated on.

The MSF Agile uses the principles of the Agile Software Development approach formulated by the Agile Alliance. It provides a process guidance which focus on the people and changes.

[183] Source: MSDN Library article. Microsoft Corporation.

[184] Sridharan, P. 2004. Visual Studio 2005 Team System: Overview.

It includes learning opportunities by using iterations and evaluations in each iteration. The MSF CMMI model provides additional formality, reviews, verification and audit. It has more mandatory documents and reports than the agile version, and this more formal development process reduces risk on large software projects and provides a measurable status.

Agile practitioners pride themselves on highly productive, responsive, low ceremony, lightweight, tacit knowledge processes with little waste, adaptive planning and frequent iterative delivery of value. It is often assumed that CMMI compliant processes need to be heavyweight, bureaucratic, slow moving, high ceremony and plan driven. Agile developers often sceptically perceive formal process improvement initiatives as management generated inefficiency that gets in the way of productivity. Anderson and his team at Microsoft adopted the teachings of W. Edwards Deming and stretched the MSF for Agile Software Development method to fit the requirements for CMMI Maturity Level 3. The resultant MSF for CMMI Process Improvement is a highly iterative, adaptive planning method, light on documentation, and heavily automated through tooling. It enables management and organisation of software engineering through use of agile metrics such as velocity and cumulative flow but with an added dimension of an understanding of variation – adapted from Deming's teachings.[185]

Hamel and Highsmith[186] recommend that, as with the ISO standards, the CMMI process should be tailored for the organisation and then tailored for each project. Few achieve CMMI Level 5, not only because they lack the resources for continuous process improvement, but because they have a false impression of what a software process is. To reach a CMMI level, an organisation does not have to do everything described in each of the Key Process Areas (KPA) for that level; however, the goals of the KPAs must be satisfied.

Scrum for Team System[187] is a free Agile Software Development Methodology add-in for Microsoft Visual Studio Team System, developed by Conchango, in collaboration with the Scrum co-inventor, Ken Schwaber, and the Microsoft Technology Centre UK. It provides development teams with deep support for the use of Scrum, when running projects using Visual Studio Team System's integrated suite of lifecycle tools.

[185] Anderson, David J. 2005. Stretching Agile to fit CMMI Level 3 - the story of creating MSF for CMMI Process Improvement at Microsoft Corporation.

[186] Hamel, S. Highsmith, J. 2000. Optimize — or Adapt. Software Development.

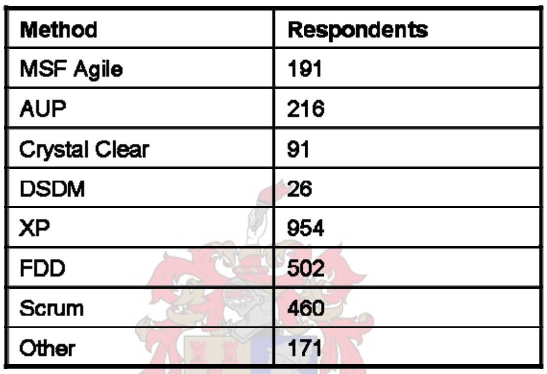
[187] See www.scrumforteamsystem.com

2.11 Adoption and comparison

Ambler's survey[188] on Agile SDLC adoption in March 2006 showed that:

- 65 percent have adopted one or more Agile development techniques.
- 41 percent have adopted one or more Agile methodologies.
- 60 percent report *increased productivity.*
- 66 percent report *increased quality.*
- 58 percent report *improved stakeholder satisfaction.*

The survey respondents ranged from small teams with less than ten members to large IT organisation with thousands of developers.



Method	Respondents
MSF Agile	191
AUP	216
Crystal Clear	91
DSDM	26
XP	954
FDD	502
Scrum	460
Other	171

Table 2-2: Adoption of agile techniques (multiple answers allowed)

Table 2-2 indicated that the most popular methodologies are XP by far, followed by FDD and Scrum. It took three decades since Brooks' initial cries to have reached this milestone. This is a short period in human lifetime and a long period in computer lifetime. Nevertheless the milestone marks significant progress regarding our understanding of efficiency when it comes to the development of computerised systems. Even though Brooks identified the same difficulties that are still plaguing the practice today, he did not offer any concrete solutions at the time. It has now come to pass that what originally seemed insipidly unprofessional at the turn of the millennium, has now unequivocally been proven to contain components of a Silver Bullet for the management and practice of System Development Life Cycles.

[188] Ambler, S.W. 2006. Survey Says Agile Works in Practice.

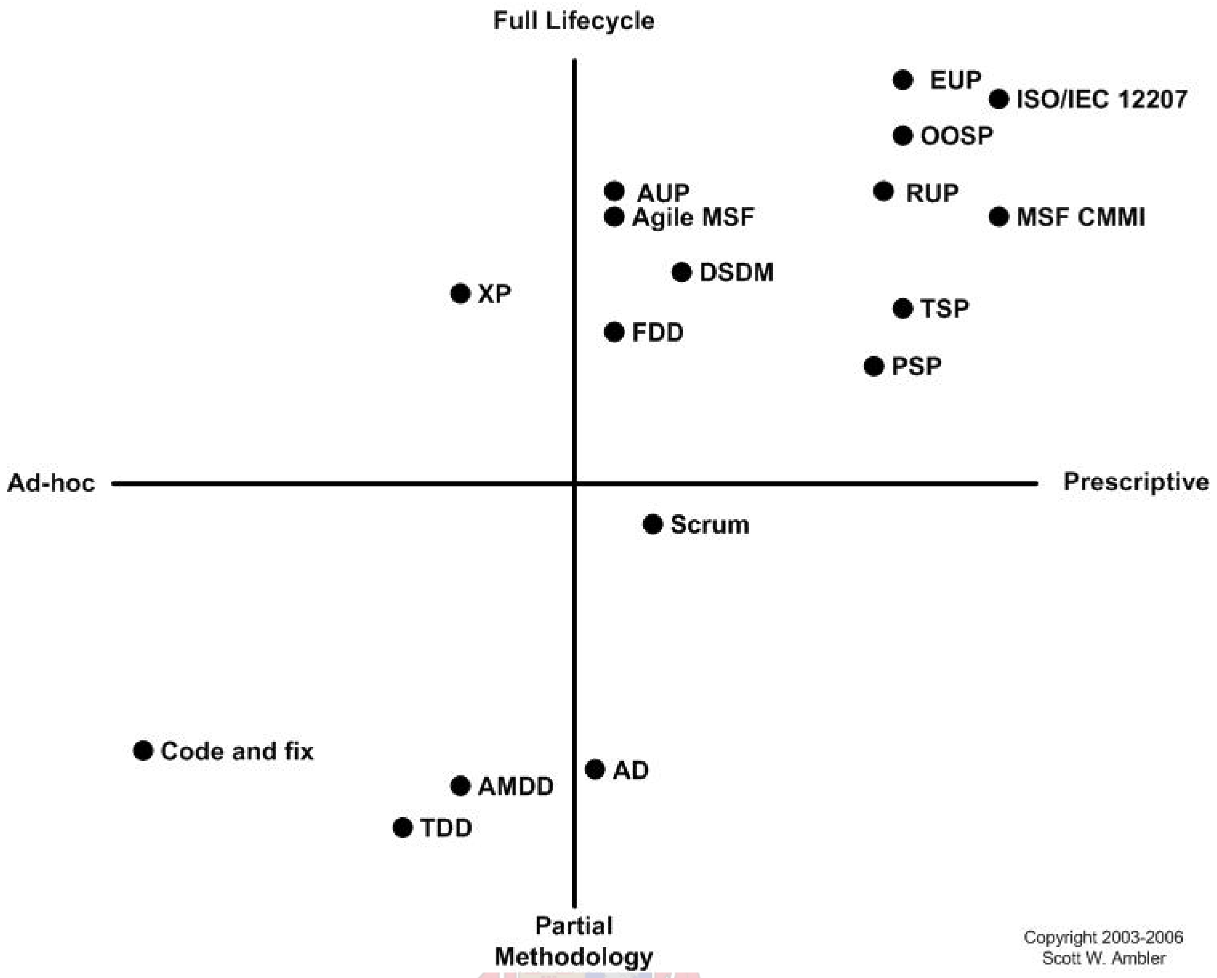


Figure 2-11: Comparison of methodologies

Ambler[189] provides various comparisons of most of the known methodologies as illustrated in Figure 2-11. All the development of prescriptive and complete methodology standards were moving further and further away from the evasive Silver Bullet.

2.12 Conclusion

Traditional software development methodologies are based on a closed-world assumption that the boundary between system and environment is well defined and static. This assumption is flawed. Software systems need to keep pace with its unpredictable open-world environment allowing it to quickly react to changes by self-organising its structure and self-adapting its behaviour.[190] The problem domain is therefore not bound by the methodologies of software construction alone. It has to address the architectural domain of how software systems must be designed to endure change.

[189] Ambler, S.W. 2006. Choose the right software method for the job.

[190] Baresi, L. Di Nitto, E. Ghezzi, C. 2006. Toward Open-World Software: Issues and Challenges. IEEE Computer. Vol.39. No.10.

Chapter 3
Proposed theoretical models

The more precisely the position is determined,
the less precisely the momentum is known.
The path comes into existence only when we observe it.
Werner Heisenberg, 1927

This chapter expands on the various organisational and knowledge management theories that have had some influence on SDLC methodologies described in the previous chapter. It will focus on emergent theories and factors that would most likely have positive influence in delivering more satisfactory systems. It will be seen that systems development has less to do about the specifics of the systems being developed, but more to do with the management of the workforce that develop these systems. Specific attention will be given to Sensemaking and Complex Adaptive Systems theories. In the context of the thesis the predominant means of production is human capital and more specifically *teams*. It will be shown that it is essential that the development manager understand the working of teams as complex adaptive organisms.

3.1 Introduction

Albert Einstein proclaimed that a theory should be as simple as possible, but no simpler. The topic dealt with here is one of complexity. Managers can deal with chaos by forcing it to order. However when the system you need to control is complex by nature it places higher demands on management practices. The elegance and precision of physics have long been the envy of life scientists, social scientists, businesspeople, and ordinary, literate citizens. Physics has served as the model of how knowledge should be handled.[191]

Physics is both real and abstract, but at least it seems to conform to a set of universal laws commonly known as the laws of nature. These are the laws of physical things. The simple well known powerful equation derived by Einstein,

[191] Frame, J. Davidson. 2002. The New Project Management.

$$E = mc^2 \qquad \textbf{3-1}$$

reveals the stark linear contrast between energy (E) and matter (m). It hints that an enormous amount of energy can be produced from a little amount of matter. It does however not describe the dynamics and constraints of this *transformation*. Neither does it explain life and the laws of nature for living organisms. This seems to be a much more challenging but evasive frontier of scientific discovery, perhaps only to discover that

$$V = kc^2 \qquad \textbf{3-2}$$

whereas enormous value (V) can be produced from a little knowledge (k). Yet again the formula does not yield how this transformation occurs. There are no universal units of measure for knowledge or value. These dimensions are in the subjective domain.

3.1.1 Seeing

Complex systems thinking emerged as man started dancing around a fire, casting imaginative monsters on the cave walls[192] where the children sheltered, and spurred a *parallel universe of belief*. Carlos Castaneda[193] wrote a series of books on what he calls *Becoming a Man of Knowledge*. He was a scholar of anthropology at UCLA gathering information on various medicinal herbs used by the Indians in Sonora, Mexico, when he met the old Yaqui Indian on which the books are based. Castaneda writes:

> We were talking about my interest in knowledge; but, as usual, we were on two different tracks. I was referring to *academic knowledge* that transcends experience, while he was talking about *direct knowledge* of the world.
>
> Don Juan's method of teaching required an extraordinary effort on the part of the *apprentice*. In fact, the degree of participation and involvement needed was so strenuous that by the end of 1965 I had to withdraw from the apprenticeship. I can say now, with the perspective of five years that have elapsed that at the time Don Juan's teachings had begun to pose a serious threat to *my 'idea of the world'*. I had begun to lose the certainty, which all of us have, that the reality of everyday life is something we can take for granted.

[192] Morgan, G. 2001. Images of Organization:216. Morgan presents a metaphorical analysis bases on Plato's cave story in The Republic. This has common ground with Gestalt theory which explores the virtual reality creative ability of brain and mind.

[193] Castaneda, Carlos. 1971. A Separate Reality.

> Apparently in this system of knowledge there was the possibility of making a semantic difference between *seeing* and *looking* as two distinct manners of *perceiving*. 'Looking' referred to the ordinary way in which we are accustomed to perceive the world, while 'seeing' entailed a very complex process by virtue of which a man of knowledge allegedly perceived the 'essence' of the things of the world.

Castaneda refers to concepts such as *sensible interpretation*, *units of meaning* and *practitioners*, which in today's terms refers to *knowledge workers*. These statements are quite relevant to the subject of Sensemaking. When dealing with models of Complex Adaptive Systems one has to avoid *looking* too deeply and rather start *seeing* what it really is.

The dawn of the twentieth century revealed the heydays of scientific discovery and the revolt against *belief systems*. Scientific enlightenment provided a new hope in objectivity and reductionism. Science demonstrated that the mastering of control over matter and spurred a *belief* that all systems are *closed* non-living systems.[194] Be therefore forewarned to acknowledge the importance of an open systems approach that is inclusive of order and disorder, control and chaos, knowable and unknowable, expected and unexpected. As scientific knowledge increases it helps to *see* the natural order in what was previously labelled as chaos. It furthermore helps to make sense of novel events,[195] and thereby reduces complexity and entropy.[196] Scientific knowledge of cybernetics is critical to maximise the economic productivity of embedded knowledge production through innovation. All of the elements are complex and abstract. Metaphors are powerful thinking aids that lies peppered across the SDLC literature landscape. Metaphors and models help to simplify complex systems and *amplify understanding* of the various phenomena. Morgan however warns that metaphors also become *a way of not seeing*.[197]

3.1.2 Embedding knowledge and complexity reduction

Through embedding complex knowledge into innovative tools, inventors catapult future generations forward without the need for them to know everything.[198] Google's co-founder, Sergey Brin, envisions a little stylish brain plug-in appliance that will make the world's

[194] Flood, R. L. 1999. Rethinking the Fifth Discipline. Chapter 9. Towards systemic thinking.

[195] Weick. K.E., Suttcliffe, K.M. 2001. Managing the Unexpected:80.

[196] Boisot, M. 1998. Knowledge Assets:11.

[197] Morgan, G. 1997. Images of Organization:5.

[198] Cox, Brad. 1995. No Silver Bullet Revisited. Cox argues that the essential complexity of software goes away once it is properly encapsulated.

knowledge immediately available on demand.[199] Gadgets such as Apple's iPods could be the embryo of such a fantastic achievement. This is certain future reality as the cost and density of persistent storage and processor power continues to obey Moore's law; and as text-to-speech and voice recognition reach the same error rate as the average human. Personal search, tagging and electronic notebook technologies would create a localised cache of the personified Internet while keeping itself in sync through global wireless Internet access. The point here is that humans no longer have to know everything if they can embed the knowledge into these kind of tools. Every time someone drives her car, she relies on a lot of embedded knowledge that she does not need to know unless she had to build her car from raw materials each time she wanted to get somewhere. Bryson[200] reaches the same insight that humans do not yet fully know how they came to be and how every cell and organ functions, yet they *instinctively*[201] know enough to make a living. Not all toolmakers are as profound as Ford, Nokia, Google and Apple. They distribute knowledge-power[202] through the discipline of *simplicity*. It is a lot easier to succeed when the environment is designed to help you get in; get what you need; and get out.[203] Understanding data, information, knowledge, knowledge embedding and the diffusion of knowledge-power has been studied by many philosophers, psychologists, neurologists, mathematicians, physicists and computer scientists. Boisot[204] went beyond the classical epistemology and discovered that the productive use of knowledge lies in the knowledge lifecycle model what he calls the Social Learning Cycle (SLC). Only by exchanging information do people create value especially if that information leads to the conservation of energy.

3.1.3 Knowledge-power

Modern day man's preoccupation of *getting things done* is believed to be critical for survival. Like mice trapped in a rat-race business men and women live in a growing economic crisis of maximising their *busyness*[205] opposed to creating value. Robert Flood[206] draws an insightful

[199] Vise, David A. 2005. The Google Story:292.

[200] Bryson, B. 2003. A Short History of Nearly Everything.

[201] Instinctive know-how is a form of tacit knowledge that is embedded into living creatures and transferred by heredity. It is known, but not learned, and not taught.

[202] Flood, R. L. 1999. Rethinking the Fifth Discipline.

[203] Jensen, W.D. 2000. Simplicity – The New Competitive Advantage in a World of More, Better, Faster:173.

[204] Boisot, M. 1998. Knowledge Assets.

[205] Mackay, Hugh. 2005. Mind your own busyness.

[206] Flood, R. L. 1999. Rethinking the Fifth Discipline – Learning within the unknowable:95.

classification of his impression on the literature volume of books on management and organisation as illustrated in Figure 3-1.

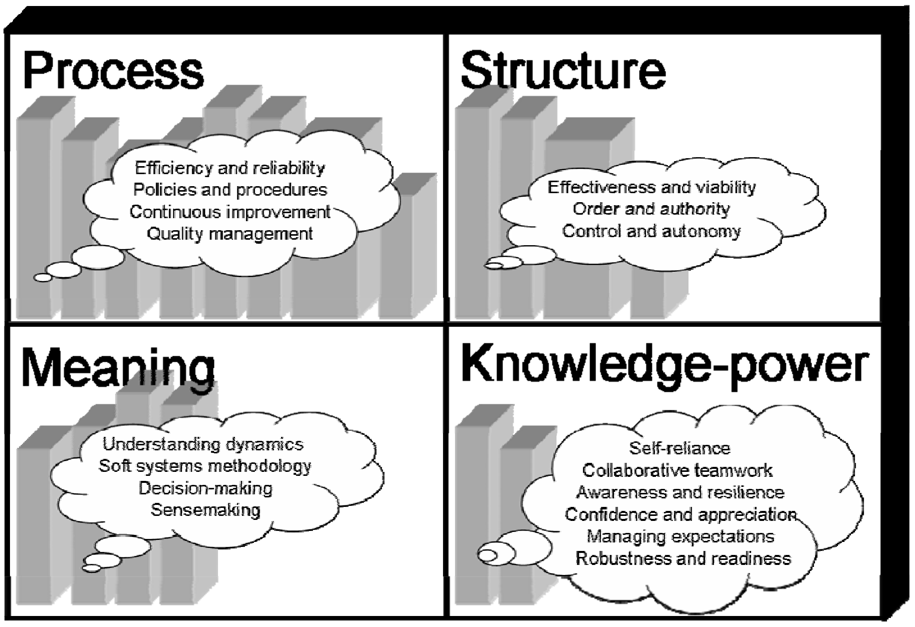


Figure 3-1: Four knowledge dimensions[207]

Flood hypothesises that the knowledge society has invested knowledge mostly on *process*, less on *structure* and *meaning,* and very little on what he labels *knowledge-power.* Knowledge-power essentially focuses on the development of *actionable self-reliant people*.

3.2 Lifecycle models

Lifecycle modelling draws developmental patterns from various life-forms, from its inception to its decay. At the physical level, a lifecycle is the sequence of developmental changes undergone by an organism from one primary form to the recurrence of a similar form in the next generation. On a sociological level, it is a series of stages that characterise the course of existence of an individual, group or culture.[208] These systems interact and transact with its environment in order to survive. It takes from the environment and gives to the environment, leaving itself and its environment in an altered state. This is its *autopoietic*[209] nature. Ultimate survival however requires subsequent mutations to include incremental and radical improvements opposed to defects. Applying this frame of reference to the context of man-made systems development, it is necessary to acknowledge all of the following *aspects and contrasts*:

[207] Adapted and synthesised from Flood's four windows and bookshelf metaphors for deepening systemic appreciation. Flood, R.L. 1999. Rethinking the Fifth Discipline. Chapter 12.

[208] Definition derived from the Random House Webster's College Dictionary. 2000.

[209] Autopoiesis was introduced by biologists Maturana and Valera in 1973.

- Economic environment and domain analysis are inseparable;
- Domain analysis and project management are inseparable;
- Project management and engineering are inseparable;
- Systems engineering and analysis are inseparable;
- Systems analysis and design are inseparable;
- Systems design and development are inseparable;
- Systems development and implementation are inseparable;
- Systems implementation and support are inseparable;
- Systems support and improvement are inseparable; and
- Systems improvement and sustainability are inseparable.

At all of these interwoven spars lie various artefacts such as strategies, principles, patterns, goals, risks, standards, practices and constraints. A small change to an artefact could have an enormous impact on the survival of the whole as defined in Complexity Theory. These critically important artefacts are created and manipulated by small teams of people or agents. They are *symbolic analysts*[210] with the explicit and tacit knowledge and skills necessary to conduct information flows and create innovative systems. *Cybernetics*[211] embroiders all these elements together into a protean model known as the Systems Development Life Cycle (SDLC).

The word *organisation* comes from the Greek word *organon*, meaning a tool or instrument.[212] Organisation management experts such as Senge, Nonaka, Boisot, Morgan, Weick, Snowden, and Marchand provide numerous *instruments* for perceiving data. The data is *conceptually* and *perceptually* filtered in context of the environment. Managers use these tools or instruments to amplify and accelerate their understanding of the data in order to take good decisions that lead to desired outcome. Figure 3-2 shows Boisot's Agent-in-the-World model that provides a comprehensible framework for understanding cybernetics and the dynamics of *Sensemaking*.

[210] Robert Reich used the term *symbolic analysts* to classify well educated people who can earn very good wages in the global market.

[211] Norbert Weiner coined Cybernetics in 1947 as the science of control and communication in and between animal and machine. Norbert Wiener. 1954. The Human use of Human Beings:16.

[212] Morgan, G. 1997. Images of Organization:15

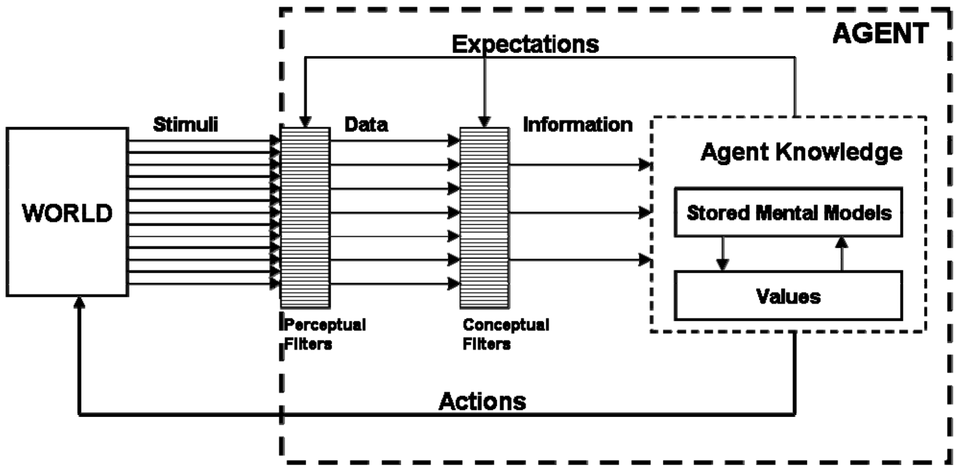


Figure 3-2: Boisot's Agent-in-the-World model[213]

The first chapter describes software development as being more of a skilled trade than an exact science.[214] Constructing complicated systems is however a complex process that needs to be executed with competitive agility. When such work needs to be performed quickly, it must mostly rely on *intuition*[215]. How does one characterise, model and represent intuition and common sense in a productive framework that is scientifically defensible? How does one solve the Software Developer's Dilemma?

The Management Consultants and Business Analysts fields have grown tremendously to close this expansive gap. These analysts are responsible for knowledge acquisition, analysis, codification and transfer. This process is most accurately modelled by Boisot's Social Learning Cycle.[216] With the increase in computer processing power, the decrease in computing and telecommunications costs, the speed of the learning is improving. This gives hope for capturing more accurate and complete requirements.

As discovered in Chapter 2, in effective systems development models such as developed by Highsmith, Cunningham, Beck, Schwaber and Sutherland, a paradigm shift gradually emerged hinting that the silver bullet lies somewhere within what Nonaka and Takeuchi called The New New Product Development Game.

[213] BOISOT, M. CANALS, A. 2004. Data, information and knowledge: have we got it right?

[214] See paragraph 1.4.1 The Software Developer's Dilemma.

[215] Intuition is proposed here as a form of tacit knowledge that is performed with high agility.

[216] Boisot, M. 1998. Knowledge Assets.

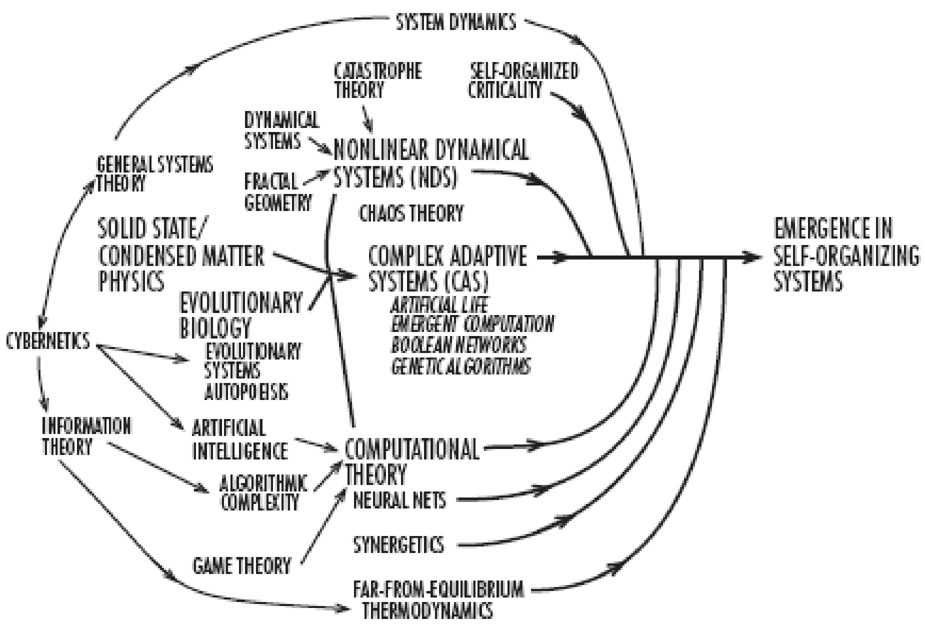


Figure 3-3: Mathematical and scientific roots of emergence[217]

The goal in this chapter is to identify and understand the characteristics of the *Self-Organising Systems Development Teams.* As illustrated in the systems diagram in Figure 3-3 these characteristics would emerge from a study of the team dynamics and behaviour with regards to the following main topics:

- *Self-management.* Setting goals, objectives and milestones;
- *Self-configuration.* Structural design, composition and dismantling;
- *Self-healing.* Discovery and counteracting of issues;
- *Self-optimisation.* Monitoring and control of resources to ensure the optimal functioning with respect to the defined objectives; and
- *Self-protection.* Securing its survival by proactive identification and protection from arbitrary attacks.

Practical and *pleasing models* are simpler to understand, simpler to communicate and simpler to implement. Simpler models do not deny chaos and complexity; it however contains, nourishes and protects it. Like the essence of blood, it forms a dynamic value creation ecosystem through free flowing cells that clog together temporally, detaches, frisky exchange of information, delivering essential resources and removing harmful barriers. Placing the product development process into a complete context it could be envisioned that an all

[217] Goldstein, J. 1999. Emergence as a Construct: History and Issues.

embracing view is considered where all the elements is transported in a dynamic free-flowing value-creating conduit as illustrated in Figure 3-4.

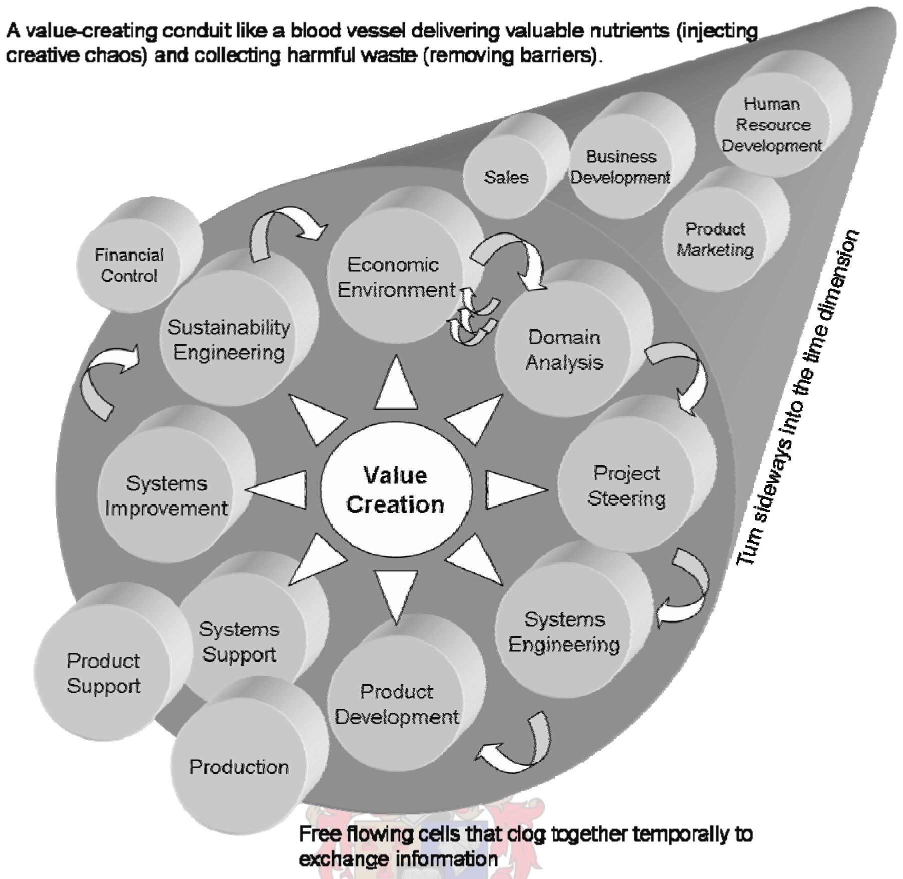

A value-creating conduit like a blood vessel delivering valuable nutrients (injecting creative chaos) and collecting harmful waste (removing barriers).
Human Resource Development
Business Development
Sales
Product Marketing
Financial Control
Economic Environment
Sustainability Engineering
Domain Analysis
Turn sideways into the time dimension
Systems Improvement
Value Creation
Project Steering
Systems Engineering
Systems Support
Product Support
Product Development
Production
Free flowing cells that clog together temporally to exchange information

Figure 3-4: A value-creating conduit for systems development

Each of these living cells is represented by teams with unique roles and goals with the context of the corporation's economic activity. Changes within each cell may have some effect on other cells and in extreme cases even small changes could have extraordinary effects. This is the predicate of chaos and complexity theory. Turning the conduit or vessel sideways presents a dynamic perspective wherein the various cells move and interact. The metaphor of blood is particularly useful in that it symbolise many of the essential attributes such as clogging, immunity, cloning, transformation, transportation, healing, supporting, adapting, and redundancy.

The goodness-of-fit will be determined between the characteristics of SDLC methodologies as described previously and that of new knowledge management theories with specific attention given to organisational sensemaking, systems thinking and cybernetics before delving more deeply into complexity theory.

3.3 Sensible leadership

What constitutes good sensible leadership? Drucker[218] argued that the *conductor of the orchestra* is probably the acme of good sensible leadership. The conductor is a highly specialised and skilled individual, works effectively with a broad spectrum of diverse and talented artists, and is hyper-focussed on the sensemaking and sense-giving cues while conducting the orchestra. Hammer[219] disagrees with the orchestra model and proposes the football team model instead. Football is played in a constant state of flux. He argues that the project strategy resembles a game plan much more than it does a musical score does. Moreover, a football team's organisation and management structure bear an uncanny resemblance to those of a process-centred company.

Drummond[220] focuses on what goes wrong with organisational decision making. She is highly critical of decision making as a science filled with technical sophistication, but devoid of feeling. Her work represents the Achilles' heel of management with the characteristics that managers should be cognisant of when leading teams. Factual data is typically devoid of intuition, feeling and sense, and is often inappropriate, incomplete, inaccurate, misleading and concealing, thereby clouding the situation by causing ambiguity and doubt. Too often numerous biases, such as anchoring, frequency, vividness and emotion, causes distortion of the reality in favour of a priori unrealistic, unachievable, and unsustainable outcomes. Managers subconsciously disguise their emotions, with false rationality with regard to planning, while faithfully going along with their intended plans. They gamble with repeated optimism and ignoring other possibilities. *The innovator's dilemma*[221] is the curse of ingenuity by getting trapped in continuity and the unfortunate trust in achieving the ultimate, but unattainable future. Managers blindly walk into these ambushes, keeping to their status quo, and naively trusting their corrupted common sense. Instead they should not accept this flawed destiny, without challenging and assessing all options.

These lessons are relevant to Phillip Su's[222] satirical story, when developers are asked for delivery time estimates. Drummond makes a very important distinction that *knowing is not the same as understanding.* Managers too often draw conclusion from what is known,

[218] Drucker, P. 1988. The Coming of the New Organization. Harvard Business Review, Vol: 66 Iss: 1 Date: Jan/Feb 1988

[219] Hammer, M. 1996. Beyond Reengineering.

[220] Drummond, H. 2001. The Art of Decision Making:81-213.

[221] Christensen, C.M. 1997. The Innovator's Dilemma. Christensen's book is entirely focussed on this trap and how to get out of it.

[222] See paragraph 1.4.4 on page 12.

without having acquired enough understanding and sense to make good decisions. Dreams, fantasy and myth-making are collectively important, because they hold the potential to create the social cohesion, the sense of belonging and commitment necessary to move people to act. That is the essence of progressive, organisational leadership.

The analysis of the extreme leadership traits of Ernest Shackleton[223] is summarised in ten leadership strategies which have direct and compelling relevance to the contemporary business world and to that of Agile Systems Development. They are:

- *Vision and quick victories*. Never lose sight of the ultimate goal, and focus energy on short-term objectives.
- *Symbolism and personal example*. Set a personal example with visible, memorable symbols and behaviour.
- *Optimism and reality*. Instil optimism and self-confidence, but stay grounded in reality.
- *Stamina*. Take care of yourself – maintain your stamina.
- *The Team Message*[224]. Constantly reinforce the team metaphor.
- *Core team values*. Minimise staff differences and insist on courtesy and mutual respect.
- *Master conflict*. Deal with anger in small doses, engage dissidents, and avoid needless power struggles.
- *Lighten up*. Find something to celebrate and something to laugh about.
- *Risk*. Be willing to take big risks.
- *Tenacious creativity*. Never give up – there is always another move.

Even though Shackleton never reached many of his ultimate goals, he instinctively new how best to lead small teams in reaching their short-term goals. He produced unprecedented productivity with his teams in the face of unforgiving circumstances.

[223] Perkins, D. N. T., Holtman, M. P., Kessler, P. R., McCarthy, C. 2000. Leading at the Edge – Leadership Lessons from the Extraordinary Saga of Shackleton's Antarctic Expedition.

[224] This is similar to the Agile use of a project *Metaphor*.

In an extensive study of more than 200 firms, Immelman[225] discovered that many of these ancient values are still common in today's organisational cultures. Immelman identified five dimensions and twenty-three attributes that describe *tribal behaviour in organisations*. Immelman's dimensions extends Maslow's motivational model by looking at individual as well as team[226] security and values. He provides a strong argument that leaders should realise that these instinctive traits can not be changed and that the secrets to effective leadership is working a strategy that cuts and moulds organisations along this grain.

Wheatley,[227] one of the original thinkers of complexity science in management, returns to the days of leaders sitting around the campfire in deep disciplined cycle of stages for solving complex problems. Her description of the five stages adds a rich colour of humility with attributes such as curiosity, patience, generosity, respect, discipline and discernment.

Richardson[228] draws further insights for the ancient Chinese philosophy based on the five seasons and five essential elements. Her advice is also that managers must work *with* the direction of the natural, ancient feedback cycles rather than against it.

Binney and Willams[229] reach the same conclusion in their synthesis of top-down and bottom-up leadership styles. Individuals shape the future by combining clear intention with respect and understanding for people and organisations. They encourage leaders to work *with* the grain, not across it, by acknowledging and supporting individuals' hopes and fears.

What metrics exist for understanding how leaders *make sense* of what conditions facilitate good performance? A model for pre-empting barriers and enablers for organisational growth, adaptation, and development can be found in Greiner's evolutionary model[230] shown in Figure 3-5.

[225] Immelman, R. 2003. Great Boss Dead Boss – How to exact the very best performance from your company and not get crucified in the process.

[226] Immelman prefers to use the word *tribe* to embrace the ancient origin of the team concept. The use of an unique *language* and set of *symbols* are all part of the tribal attributes he identified.

[227] Wheatley, M.J. 2005. Finding Our way: leadership for an uncertain time.

[228] Richardson, J. 2005. Ancient insights into the modern organization.

[229] Binney, G. Williams, C. 1996. Leaning into the Future. Changing the way people change organizations.

[230] Pries-Heje, J., Baskerville, R. L., Hansen, G. I. 2005. Strategy Models For Enabling Offshore Outsourcing.

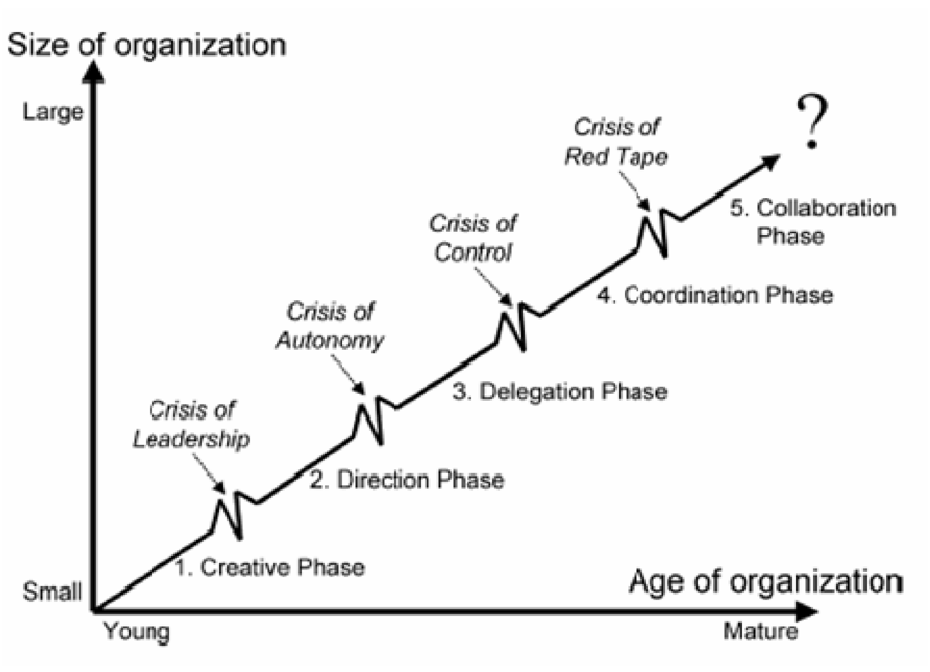


Figure 3-5: Greiner's model of organizational evolution and revolution

3.3.1 Creative phase.

The small founding team is informal, long work hours are normal, and the feedback from the market is immediate. As it grows in size and matures, the organisation reaches the *leadership crisis* when informal communication is no longer sufficient. The dedication, long hours, and small salaries are no longer sufficient motivation. New procedures are needed to exploit efficiencies of size and to provide better financial control. To solve the leadership crisis, a strong manager is needed. Often the owner or founders lack the necessary skills and knowledge, and hate to step aside even though they are unsuited to be managers. If the organisation survives the leadership crisis, it will embark on a period of sustained growth under able and directive leadership.

3.3.2 Direction phase.

Communication becomes more formal as a hierarchy is built and the upper levels take responsibility for the direction of the organisation. It is also in this phase that formalised systems for accounting, incentives, work practice, and job specialisation emerge before it hits the *autonomy crisis*. Middle-level managers see the centralised decision structure as a burden, and the more autonomous middle managers start acting independently.

3.3.3 Delegation phase.

Often top management reacts by attempting to return to centralised management. The solution to the autonomy crisis is a more decentralized organisational structure where middle managers have greater responsibility and autonomy. The delegation phase ends in the *control crisis* where top managers realise that they have lost control over a highly diversified operation.

3.3.4 Coordination phase.

The control crisis is overcome by the use of coordination techniques such as formal planning, the creation of product groups treated as *investment centres*, and by the initiation of staff functions that control and monitor leading to the crisis of *red tape*. Line managers are suspicious of staff functions and distrust grows between dispersed groups.

3.3.5 Collaboration phase.

In this last phase, strong interpersonal collaborations are established to overcome the red-tape crisis. A more flexible and behavioural approach to management is implemented through the use of teams. The staff functions are reduced in number. The motivational structure becomes more geared to team performance than to individual achievements.

Fishman[231] notes that teamwork is a harder way of doing the work, but when it clicks, the result is a seamless experience. Fishman draws this conclusion from his observations of a leading British design company, Imagination. Imagination's interdisciplinary approach puts it more on a par with a theatre troupe or a circus than with a traditional design company. The official Imagination brochure lists 26 disciplines used to attack projects -- a range of talent that gives Imagination's work its special texture. He agrees that creative people are notoriously independent and notoriously difficult to manage.[232] How does Imagination herd its extraordinary collection of talent into fast-working, high-performance teams? Their advice is as follows:

- *Start the project before there is a project.* Projects are often only loosely defined at the start. Teams are assembled early, often before the company and the client have reached a final agreement about the goals of the project. In that way, the team often defines the project, rather than the project defining the team.
- *Make the brief brief and share it.* Even for the most complicated projects, team members ultimately know exactly what the goal of the project is. All members use the same words and phrases to express that goal, and the goal is usually boiled down to a sentence or two. Every idea can be tested against what the team and the client are trying to accomplish.

[231] Fishman, C. 2000. The Total Teamwork Agenda.

[232] Berkun, S. 2005. The Art of Project Management. The notion of managing talented individuals has often been described as that of herding cats.

- *Everyone comes to the table.* Projects are managed through weekly meetings -- meetings in which ideas are batted around, problems are raised, and progress on deadlines is assessed. Involve everyone in a project by inviting everyone to all meetings. Production people and client-contact people are just as much a part of the team as creative types. The result reduces production problems, and client-service representatives have the information that they need to keep clients informed and satisfied.
- *Disperse responsibility.* On project teams, no one is actually in charge. The result is not chaos, but just the opposite. Dispersing the power also disperses the responsibility.

The various approaches described here and in the preceding paragraphs correlates very well with many of the Agile Development Methodology principles such as project vision, metaphor, customer on-site, open face-to-face communication and self-organising teams.

3.4 Sensemaking in Organisations

Leedom[233] noted that research must build from qualitative description toward quantitative prediction of performance, using a range of investigation methods such as:

- Observation (non-intrusive);
- Subjective investigation (ethnography, knowledge elicitation);
- Storytelling and anecdotes (knowledge building);
- Metaphor (pattern matching);
- Scientific method (controlled, empirical hypothesis testing); and
- Mathematical analysis (baseline modelling, sensitivity analysis).

Weick and Sutcliffe[234] elaborate on the *five qualities of mindfulness* and the organisational processes and leadership practices that contribute to a mindful infrastructure. The five qualities are grouped under two broad headings. This first is *anticipating the unexpected* by having a preoccupation with failure, reluctance to simplify, and sensitivity to operations. The second grouped as *containing the unexpected* when it occurs by having a commitment to resilience, and deference to expertise. Teams that have these capabilities counteract traps

[233] Leedom, D. K. 2001. Sensemaking Symposium Final Report.

[234] Weick, K.E. Sutcliffe. K.M. 2001. Managing the Unexpected.

that are built into expectations, detect the unexpected sooner, contain the unexpected more fully, and learn from these local and contingent responses.

Weick and Sutcliffe further claim that plans and operating procedures have effects that run exactly counter to the processes of mindfulness. Plans, for example, embody expectations and thus narrow perceptions by reducing the range of things that people notice. The typical organisation's emphasis on routine and contingency planning embodies assumptions that weaken the ability to respond to the unexpected and foster new learning. This is the antithesis to the processes of mindfulness essential to achieving reliable outcomes in an increasingly complex and volatile world. The reason these qualities are not more visible and influential is that most organisations look for lessons on how to survive from organisations like themselves. They should look instead to organisations that, on the surface, look quite different; high reliability organisations that have, of necessity, learned how to manage the unexpected.

Weick provides the following properties of Sensemaking:[235]

- *Grounded in identity construction*. Making sense of the environment influences, and is influenced by one's self-concept and personal identity.
- *Retrospective*. Making sense of the present is always grounded in past experience, including past decisions to adopt certain plans and goals.
- *Enactive of sensible environment*. Making sense involves the construction of reality by assigning authority to events and cues vis-à-vis a specific context, activity, or ontology.
- *Social*. Making sense involves the creation of shared meaning and shared experience that guides organisational decision-making.
- *Ongoing*. Making sense is a continual process of refining understanding, taking action, and restoring equilibrium within the context of a specific project.
- *Focused on and by extracted cues*. Sensemaking involves the process of people noticing and extracting specific cues from the environment and then contextually interpreting those cues according to certain held beliefs, mental models, rules, procedures, stories, and so forth.

[235] Weick, K.E. 1995. Sensemaking in Organizations.

- *Driven by plausibility rather than accuracy*. Sensemaking is driven by the need for a workable level of understanding that guides action, rather than by a search for universal truth.

Jelinek[236] noted that traditional organisations are designed to produce stable, predictable performance by eliminating ambiguity and unauthorised behaviour. Such organisations use task decomposition and specialisation to narrow participant focus; a practice that is common in most command and control organisations. Emphasis is placed on control and managerial intent, while other *cognitive resources are generally ignored.* Attention tends to be limited to the managers of such organisations; the result being consistency and rigidity of thinking. By contrast, these same organisations do not respond well to crisis and ambiguity. If such organisations are to successfully adapt, they must organise for innovation by emphasising organisational change and learning, facilitating shared cognition, and embracing ambiguity as opportunity.

Addressing cognition and decision-making from an overall systems perspective, Jelinek has identified three cognitive elements that contribute to the ability of organisations to respond effectively to crisis and ambiguity. These elements include:

- *Shared management*. Everyone in the organisation (down to the lowest levels) is responsible for overall system performance.
- *Mindful alertness to anomalies*. Because data takes on meaning only in context, subordinates should be alert to patterns, anomalies, and change and push this information upward in the organisation.
- *Ambiguity absorption*. Organisational design should attend to who deals with ambiguity in the organisation, how data is matched up with those who provide context and interpretation, what are the attentional resources within the organisation, and where does there need to be shared interpretation.

Jelinek therefore puts emphasis on data-based organisations that focus on real causes and real results, that emphasise learning and improvement, that facilitate information sharing in order to empower all participants, and that require decision-makers to *listen down* to subordinates who have more direct access to situation awareness of the environment.

[236] Leedom, D. K. 2001. Sensemaking Symposium Final Report.

The concept of sensemaking is difficult to fully comprehend. Weick presents several anecdotes and stories to ease the learning, but the theory demands a fundamental shift in management's perception of organisational learning.

It is an important topic and it is important to read as many stories on how good leaders make sense of their situations. Stories about self-deception[237] are another important aspect of sensemaking that is cancerous towards organisational performance if not counteracted. Storytelling is an effective tool to break up fossilised mental pathways and establish new frames for holistic *systems thinking*.

3.5 Systems thinking

Senge[238] popularised organisational learning through his pragmatic approach to systems thinking. Senge enumerated several management blind spots which he calls the *seven learning disabilities* such as people being tied to their positions, blaming external factors, waiting for others to take decisions, dominated by recent events, not seeing gradual changes[239], reliance on experience[240], and the myth of the management team[241]. Senge's work influenced several of the Agile Manifesto founders and supporters.[242] It is therefore appropriate to briefly recap on Senge's five disciplines:[243]

- *Personal mastery* is the discipline of continually clarifying and deepening personal vision, of focusing energies, of developing patience, and *seeing* reality objectively.
- *Mental models* are deeply ingrained assumptions, generalizations, and pictures that influence the understanding the world and how to take appropriate action. The discipline of working with mental models starts with learning to unearth internal models, to expose individual thinking and to open it to the influence of others.
- *Building shared vision* of the future that fosters genuine commitment and enrolment rather than compliance.
- *Team learning* starts with dialogue that draws on the capacity of members to think together. Where the intelligence of the team exceeds that of the individuals, and

[237] Arbinger Institute. 2002. Leadership and Self Deception – Getting Out of the Box.

[238] Senge, P. 1990. The Fifth Discipline.

[239] The parable of the boiled frog.

[240] This complements Weick and Sutcliffe's quality of *difference to expertise*.

[241] Chris Agryris coined this as skilled incompetence – teams of people who are incredibly proficient at keeping themselves from learning.

[242] At least, Anderson, Highsmith, and Sutherland cited Senge.

[243] Senge, P. 1990. The Fifth Discipline.

where they develop extraordinary capacities for coordinates action and exceptional productivity.

- *Systems thinking* is a conceptual framework, a body of knowledge and tools that has been developed to make full patterns clearer and to help *seeing* how to change them effectively.

Senge[244] defines *participative openness* as the ability of speaking out; while *reflective openness* is the willingness to challenge ones own thinking by developing the skills of inquiry, reflection, and dialogue. People learn most rapidly when they have a genuine sense of responsibility for their actions. Helplessness, the belief that people cannot influence the circumstances under which they live, undermines the incentive to learn, as does the belief that someone somewhere else dictates their actions. Conversely, if they know their fate is in their own hands, then their learning matters. Senge defines the metaphor of localness as being the ability to decentralise control to localised organisations where it matters most. Localness thus unleashes people's commitment by giving them the freedom to act, to try out their own ideas and be responsible for producing results.

Flowchart style designs address what Senge refers to as *detail complexity*. Detail complexity arises due to the sheer volume of tasks to be done and is the forte of classical management practices. However with *dynamic complexity* the variables of cause and effect are subtle and not obviously noted over a time period. To him the real leverage in most management situations lies in understanding dynamic complexity, not detail complexity. He uses Systems Diagrams to model dynamic complexity with positive and negative influences, and delays.

Wells' models[245] for explaining Extreme Programming address detail complexity by showing the possible multi-path transitions between *inside-the-box* processes. Scrum[246] on the other hand addressed dynamic complexity with a much simpler *outside-the-box* model of monthly cycles and daily cycles.

Once the systems thinking models are defined, then the need is to learn how to effectively manage according to these models. This is the domain of Cybernetics.

[244] Senge, P. 1990. The Fifth Discipline:277.

[245] See Figure 2-5, Figure 2-6 and Figure 2-7.

[246] See Figure 2-8: Scrum Development Process.

3.6 Cybernetics in Organisations

Cybernetics[247] enhances understanding of communication and control theories. It is concerned with directed information flow in complex systems. Although the practice of cybernetics originated in nautical navigation, and its scientific use was primarily applied to mechanical and electrical engineering problems, its model of feedback, control, and regulation has also proven to be valuable to the understanding of biological and social systems. The example of the helmsman maintaining a course towards a goal, as illustrated in Figure 3-6 provides a practical yet accurate model of the role and function of the team leader or Scrum Master in the context of Agile Systems Development.

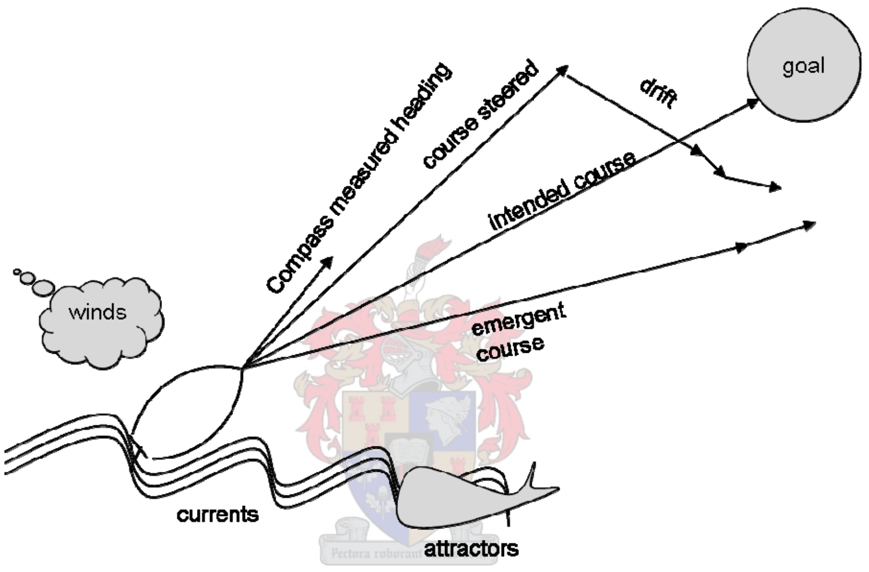


Figure 3-6: Cybernetics and emergence

Setting oneself on a predetermined course in unknown waters is the perfect way to sail straight into an iceberg.[248] Mintzberg argues that it is dangerous to articulate strategies because *explicit strategies* are blinders designed to focus direction and block out peripheral vision. This argument relates with Schwaber's[249] discussion of two types of systems, the first is the *envisioned system*, a system as initially foreseen and described by customers to deliver needed business value, and the second is the *essential system*, a system with that minimum set of functionality, architecture and design that delivers the envisioned system's business value. Business value is defined as capability that provides business advantage for a certain cost delivered by a specified date with adequate quality.

[247] Cybernetics is a word stems from the Greek *kubernetes*, or helmsman, and relates to the control and feedback behaviour required for steering or piloting a vessel.

[248] Mintzberg, H. 1987. The Strategy Concept II: Another Look at Why Organizations Need Strategies.

[249] Schwaber, K. 2001. Agile Software Development with SCRUM.

The envisioned system is a starting point for a development project. However, it is not a sufficient or correct description of the system that will deliver the business value. One cause of insufficiency is that the business environment changes during the project lifespan. A system that would provide business value at the start of the project often is insufficient by the end of the project. Another reason is communications. When the development team translates the envisioned system into working software, misunderstandings and lack of knowledge cause inaccuracies. Applying the helmsman metaphor, the effects of currents, crosswinds, compass error and strange attractors leads to a certain amount of drift or deviation.

Still another reason the envisioned system is insufficient is that customers change their minds. Schwaber[250] defines Scrum's Uncertainty Principle as, "customers don't know what they want until they see it, and they always reserve the right to change their mind." When the customer sees the envisioned system actually working, they often have different ideas about how this functionality could have best delivered the business value.

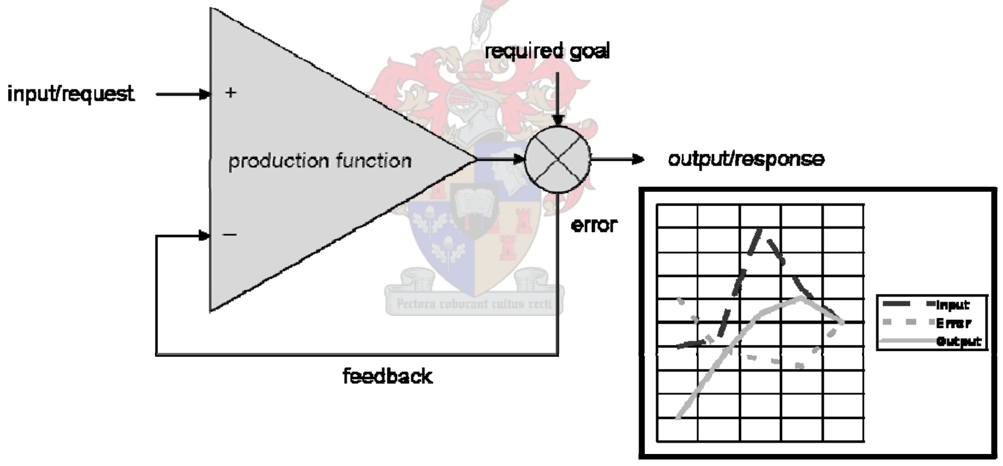


Figure 3-7: An example of a Cybernetic System

The example of a simple control system in Figure 3-7 illustrates how the response reacts to the changing request using negative feedback as commonly applied in control systems theory. A healthy system would exhibit *efficiency* and *effectiveness*. Efficiency in that it reacts responsively and effectiveness in that it conserves energy in reaching the goal. A practical and common example of a control system is the everyday elevator. As an elevator pod approaches the desired level it is most efficient and effective when it reaches its precise position in the shortest time without vacillating.

[250] Schwaber, K. 2001. Agile Software Development with SCRUM.

Kast and Rosenzweig[251] identified three dimensions of organisational performance as being effectiveness, efficiency, and *participant satisfaction*. An explanation of their third dimension in the context of the elevator example is that the customer may have not explicitly stated how pleasantly the elevator should function, but would have focussed on more quantitative metrics such as number of levels and load capacity. Engineers and customers reach satisfaction when explicit and implicit goals are achieved. The key point is that agile approaches plan for features (not tasks) as the first priority because features are what customers understand.[252]

It is possible to be effective but inefficient, thus squandering human and material resources. Similarly, it is possible to be efficient and ineffective. Peter Drucker often said that organisations sometimes emphasise doing things right at the expense of doing the right things. An important relationship is that good task performance typically leads to satisfaction so that people can work on participant satisfaction by being both effective and efficient.

Maintaining the balance between efficiency, effectiveness and satisfaction requires agile complex adaptive behaviour. Highsmith[253] supports a world view that organisations are Complex Adaptive Systems, stating that decentralised, independent individuals interact in self-organising ways, guided by a set of simple generative rules, to create innovative emergent results.

3.7 Complex Adaptive Systems in Organisations

Coleman[254] defines complexity theory in organisations as *Complex Adaptive Systems* that co-evolve with the environment through the self-organising behaviour of agents navigating *fitness landscapes* of market opportunities and competitive dynamics.

Stacey's[255] study of the complex response processes in organisations is modelled after that of Complex Adaptive Systems where individuals in a group communicate through *gestures* to elicit a desired response within the collaborative organisation or group. Gestures are symbolic representations of meaning. Stacey is a strong proponent that tacit knowledge cannot be transformed into explicit knowledge and in the context of Complex Adaptive Systems gestures is the only effective means of organisational communication.

---

[251] Kast, F. E., Rosenzweig, J. E. 1970. Organization and Management – A systems approach: 21.

[252] This explains the popularity of Coad's Feature Driven Development (FDD) methodology as evident in the survey results shown in paragraph 2.11on page 72.

[253] Highsmith, J. Cockburn, A. 2001. Agile Software Development: The Business of Innovation.

[254] Coleman, H.J.Jr. 1999. What Enables Self-Organizing Behavior in Businesses.

[255] Stacey, R. D. 2001. Complex Responsive Processes in Organizations – Learning and knowledge creation.

Phelan[256] describes complexity science as simple causes for complex effects. At the core of complexity science is the assumption that complexity in the world arises from simple rules. Generative rules typically determine how a set of agents will behave in their environment over time, but it can not predict an outcome for every state of the world. Instead, generative rules use feedback and learning algorithms to enable the agent to adapt to its environment over time. The application of these generative rules to a large population of agents leads to emergent behaviour that may bear some resemblance to real world phenomena. Finding a set of generative rules that can mimic real world behaviour may help researchers predict, control, and explain hitherto unfathomable systems.

Kurtz and Snowden[257] believe that the modelling of complex systems are valuable tools in certain contexts, but are of more limited applicability when it comes to managing people and knowledge. They identified at least three important contextual differences between human organisations and those of natural Complex Adaptive Systems such as ant colonies. These differences make it significantly more difficult to simulate these systems using computer models. The differences are:

- Humans are not limited to one identity;
- Humans are not limited to acting in accordance with predetermined rules; and
- Humans are not limited to acting on local patterns.

Complexity theory is a way of explaining how patterns emerge through the interaction of many agents. There are cause and effect relationships between the agents, but both the number of agents and the number of relationships defy categorisation or analytic techniques. Emergent patterns can be perceived but not predicted. This phenomenon is known as *retrospective coherence*. Structured methods appose retrospectively coherent patterns and codifying them into procedures will only elicit new and different patterns for which the system is ill prepared. Once a pattern has stabilised, its path appears logical, but it is only one of many that could have stabilised, each of which would have also appeared logical in retrospect. Patterns may repeat for a time, but one can never be sure that they will continue to repeat, because the underlying sources of the patterns are not open to inspection (and observation of the system may itself disrupt the patterns). Thus relying on expert opinions

[256] Phelan, S.E. 1999. What is complexity science, really? Emergence. A Journal of Complexity Issues in Organizations and Management. The New England Complex Systems Institute.

[257] Kurtz, C. F. Snowden, D. J. 2003. The new dynamics of strategy: Sense-making in a complex and complicated world.

based on historically stable patterns of meaning will insufficiently prepare managers to recognise and act upon unexpected patterns.[258]

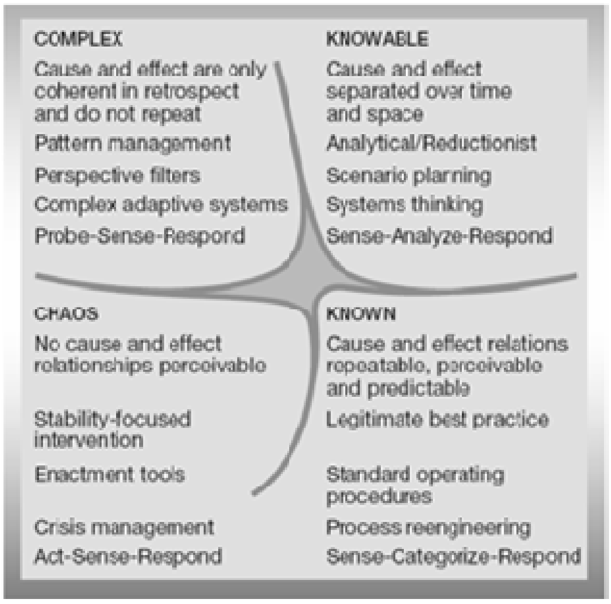


Figure 3-8: Cynefin sensemaking framework[259]

Pelrine[260] claims that people do not make rational decisions. The human brain evolved to make first fit (not best fit) pattern matches with prior experience and then retrospectively *justify them as rational*. For him this is not the way to run a systems development team effort. This fact means that you either have to convey a new message in such a way that it *resonates* with an existing prior pattern of success, or disrupt those patterns so that people see things from a different perspective, with a disposition to act. The Cynefin[261] sensemaking framework show in Figure 3-8 provides an unbiased, pre-hypothetical basis for analysing situations, issues and problems, and serves as a basis for discovering novel, oftentimes optimal solutions to them. Pelrine applies the Cynefin framework for problem-solving in the Agile Systems Development domain.

Flood[262] makes a similar argument regarding the nature of what he labels *human systems*. Human systems are not ultimately predictable and cannot be dealt with in any commonly

[258] Kurtz, C. F. Snowden, D. J. 2003. The new dynamics of strategy: Sense-making in a complex and complicated world.

[259] Kurtz, C. F. Snowden, D. J. 2003. The new dynamics of strategy: Sense-making in a complex and complicated world.

[260] Pelrine, J. 2006. Cynefin - Making Sense of Agile.

[261] *Cynefin* is a Welsh word analogous to *heritage* but with a wider scope, essentially "everything which makes us what we are". The Cynefin framework was developed by Kurtz and Snowden at the IBM Cynefin Centre for Organisational Complexity. In 2006 it was renamed Cognitive Edge <http://www.cognitive-edge.com>

[262] Flood, R. L. 1999. Rethinking the Fifth Discipline:87.

used sense of the term predict-and-control. People are not supreme planners and masters over their own lives or anybody else's. Complexity is the source of great uncertainty that mainly prevents this. Flood thereby confirms the effectiveness of Scrum's daily meetings to resolve issues by stating that the management of interrelated issues is considered to be far more relevant than any other problem-solving technique. Problems, issues and dilemmas is however inherently recurring. Systemic awareness leads to the importance of understanding of boundary judgements at the edge between chaos and complexity. Leaders need to develop formal social teams and harness energy from its members through recurring spontaneous self-organising that generates novelty and creativity in a managed dynamic full of tension. This is fully embodied within Scrum.

Highsmith[263] believes that the sweet spot for agile practices lies in the *exploratory* projects category. He further believes that there are increasing levels of unpredictability in the turbulent economy and that the goal of repeatable processes is unattainable. He agrees with Dee Hock that a *chaordic*[264] style of adaptive management is needed for creating an ecosystem with the *requisite variety* to meet the challenges of extreme projects that exhibit high change. Highsmith advocates that one needs to seek a level of *barely sufficient* prescriptive processes. The desirable objective is to execute a systems development project that:

- focuses and delivers the essential system only, since anything more is extra cost and maintenance;
- takes into account that the content of the essential system may change during the course of the project because of changes in business conditions;
- allows the customer to frequently view working functionality, recommend changes, and have changes incorporated into the system as it is built; and
- delivers business value at the price and cost defined as appropriate by the customer.

The customer steers the cost, date, and business value continuously. By increasing the cost, the customer can cause the delivery of business value sooner. By changing priorities in the product backlog, the customer can change the order in which business value is created.

[263] Highsmith, J. 2002. What is Agile Software Development? CrossTalk.

[264] Hock, D. 2000. Back to Nature. Hock defines *Chaordic* as 1) the behaviour of any self-governing organism, organization or system that harmoniously blends characteristics of chaos and order; 2) patterned in a way dominated by neither chaos nor order; 3) characteristic of the fundamental organisational principles of evolution and nature.

Highsmith[265] further recognised that while emergence is the most important part of Complex Adaptive Systems theory from a management perspective, and the need for adaptive systems development arises when there are many independent agents such as developers, customers, suppliers, and competitors, all interacting with each other, fast enough those linear cause-and-effect rules are no longer sufficient for success. Highmith eloquently remarks that:

Size and technological complexity are less important factors and planning is a paradox in a complex environment where following a plan produces the product you intended, just not the product you need.

He concludes that if Microsoft had succumbed to deterministic quality measures, it probably would not survive as they would fail to meet the demands of an unstable, complex and messy world.

Closed systems cannot evolve and is defeated by its inherent entropy. Autopoietic systems, on the other hand, have the ability to evolve and make use of entropy to grow. Entropy is the only quantity in the physical sciences that *picks* a particular direction for time, sometimes called an arrow of time. Moving *forward* in time, the Second Law of Thermodynamics states that the entropy of an isolated system can only increase or remain the same; it cannot decrease. The equation for entropy as defined by Rudolf Clausius uses the symbol S after the Greek word for *transformation*.[266] The equation for entropy is:

$$\partial S = \frac{\partial Q}{T} \qquad \textbf{3-3}$$

where Q is the amount of heat absorbed at an absolute temperature T. Two laws therefore describe the energetic state of a system. The first law states that energy is conserved. The second law states that in a closed system entropy increases until a state of equilibrium is reached for a particular transfer of heat into the environment.

Society and technology base its developmental prosperity on the scientific discoveries and explanations of the various world phenomena. Boisot[267] recognised that these discoveries emerge in what he calls the Chaotic Regime. Each discovery is based on the gradual, practical research focusing on the preceding discoveries. Boisot's *Evolutionary Production Function,* as shown in Figure 3-9, can therefore be explained in the following terms:

---

[265] Highsmith, J. 1997. Messy, Exciting, and Anxiety-ridden: Adaptive Software Development.

[266] Gillispie, C.C. 1960. The Edge of Objectivity.

[267] Boisot, M. 1998. Knowledge Assets: Chapter 4.

- Energy cannot be created or destroyed.[268]
- Knowledge can however be created and destroyed.[269]

The gradual upswing of the curve is based on natural creative chaos or entropy. New discovery occurs in an instant and therefore reduce the complexity for society at large by providing a new model or framework of understanding of complex phenomena.

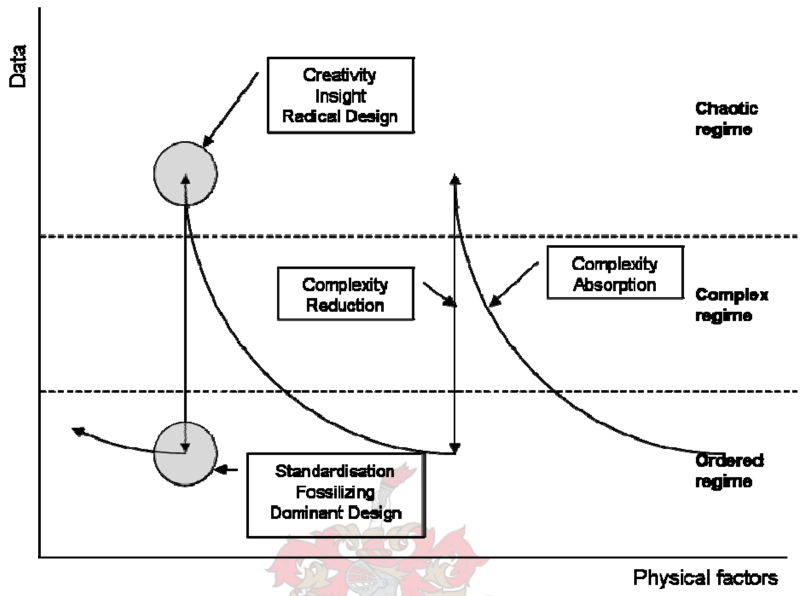


Figure 3-9: Boisot's Evolutionary Production Function[270]

Boisot claims that knowledge and entropy production stand in inverse relationship to each other. Thus by creating new knowledge one reduces entropy in some way. In this context entropy is to be associated with the unexpected or unknown order, or as Kurtz and Snowden[271] calls it *unorder*. This is also what Weick and Sutcliffe[272] argues in terms of preparing for the unexpected. Boisot argues that organisations operating predominantly in the Complex Regime and at the *edge of chaos*[273] would require greater data processing capacities for its effective management. In the context of system development teams, entropy is the creative force. Entropy can therefore be equated with creativity. Creativity has a forward arrow that seeks to grow in an open system and evens out in a closed system.

[268] This is based on the well known laws of thermodynamics.

[269] For example, the world is no longer flat, but it also is not exactly round either.

[270] Boisot, M. 1998. Knowledge Assets.

[271] Kurtz, C. F. Snowden, D. J. 2003. The new dynamics of strategy: Sense-making in a complex and complicated world.

[272] Weick, K.E., Sutcliffe, K.M. 2001. Managing the Unexpected – Assuring High Performance in an Age of Complexity.

[273] Kaufmann, S.A. 1993. The Origins of Order. Self-Organization and Selection in Evolution.

Boisot[274] recommends having an *entropy budget* whereby the rate of entropy production is kept at a sustainable level in order to survive as a *Complex Adaptive Enterprise*.

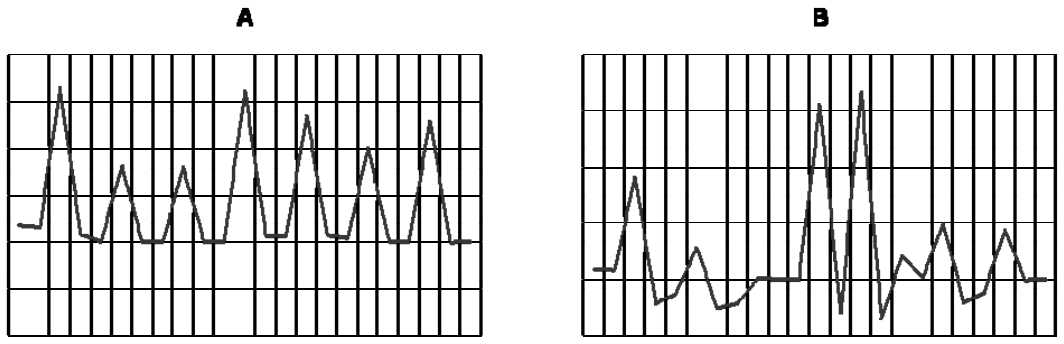


Figure 3-10: Typical variable control charts

In Figure 3-10, Meyer and Davis[275] illustrate two graphs that resemble Six-Sigma-like variance charts. Chart A looks much more predictable and stable compared to chart B. The graphs are heart-rate charts and patient A died eight days later, while the heart of patient B is a typical healthy heart that adapts to inputs from its environment. Tight control and stability is not a trait for survival. During their analysis of what makes systems alive, Meyer and Davis found the following principles of an adaptive enterprise:

- *Self-organise.* Create a community of contributors for product development; manage the rules, not the people; establish rules for people that enable flexible processes and drive adaptive behaviour.
- *Recombine*. Use reusable modules and standards to rapidly refine and customise products; seek diversity and encourage free, frequent interaction among people, partners, and communities.
- *Sense and respond.* Install feedbacks loop in every product and service for real-time maintenance and upgrade information; create markets for talent.
- *Learn and adapt*. Establish institutional learning mechanisms; exploit the learning value of failure; make knowledge management work.
- *Seed, select, and amplify*. Actively test diverse options and roll out the winners; use agent-based simulations to test rules and governance structures; keep your line-up fresh by introducing new people often.

[274] Boisot, M. 1998. Knowledge Assets:16.

[275] Meyer, C. Davis. S. 2003. It's Alive – The coming convergence of information, biology and business:.216

- *Destabilise*. Exploit the opportunities of short product lifecycles; establish a policy of turnover to continually refresh the idea pool.
- *Monetising molecules*. Look for opportunities of physical transformation in your business, searching for improvements to reduce costs and add value by shrinking mass.

Underneath these concepts, there is a lot of overlap with what is known from the Agile principles. The molecular metaphor maps well to the principle of maintaining small teams and frequent small releases. Similarly, Morgan[276] defines the brain as metaphor for a viable organisation. He defines five principles of holographic design to be:

- *Fractal*. Build the whole into the parts. Establish the vision, values, and culture as a recursive corporate code. Establish a networked intelligence with structures that reproduce itself into holistic yet diversified teams.
- *Redundancy*. Build redundancy into information processing, skills and work. Leverage equifinality[277] to achieve optimal performance.
- *Requisite variety*. Internal complexity must match that of the environment through continuous differentiation and integration. Avoid silos and atrophy by responsively adapting and becoming agile.
- *Minimum specifications*. Define no more than is absolutely necessary.
- *Double-loop learning*. Learning to learn. Scan and anticipate environmental change. Leverage emergence.

Organisational systems designed according to these principles are highly robust and fit for survival. Many living systems such as ant colonies, bees, butterflies, fish, penguins and dolphins have survived in this way. These are Complex Adaptive Systems which Morgan studies from in his *Flux and Transformation* metaphor. He cites an example of a *self-organising team* involved in the development of a new product and as well as an autonomous team in a *Just In Time* flexible factory. Morgan[278] goes on to emphasise that the fundamental role of managers should be to shape and create contexts in which appropriate forms of self-

[276] Morgan, G. 1997. Images of Organization:102.

[277] Morgan, G. 1997. Images of Organization:41. This principle defined that there will be many ways to reach a given state.

[278] Morgan, G. 1997. Images of Organization:267-269.

organisation can occur. Morgan uses the Lorenz attractor as a metaphor for managing change by creating instability that will help new attractor patterns of behaviour to emerge.

According to Stacey,[279] Complex Adaptive Systems requires a large number of individuals to resonate and cause emergence. However in the context of teams the team productivity starts breaking down when the size exceeds double digits. This is however not the case with Complex Adaptive Systems where critical mass and redundancy is a prerequisite for fitness and ultimate survival. With only two interconnected network nodes there is only one connection at play. With three it becomes three connections, and with four nodes it shoots up to six. The mathematical equation for this interconnectivity is: [280]

$$K = N\frac{(N-1)}{2} \tag{3-4}$$

Kauffman describes it as an NK Boolean network in which N represents the number of nodes and K the average interconnectivity, assuming that some nodes may not be directly linked. This linking behaviour could be manipulated via a control parameter. Kaufmann demonstrates how the network can be made to exhibit order, chaos, or the transition between the two domains, which he calls the *edge of chaos*.[281]

Schwaber[282] recommends seven plus minus two people in a Scrum team. With seven people there are 21 interconnections. Adding one more player adds 7 more interactions. Adding two more adds 15 more interactions. In classical management this complexity is managed quite simply by breaking the whole into smaller teams and thereby creating a hierarchical order. In traditional management systems communication is passed down the ranks introducing delays and corruption. However, in Complex Adaptive Systems communications happen in *parallel* and is *synchronised*.

Sutherland[283] recently published results on exceptional productivity increases by establishing a multi-national geographically dispersed large scale project employing a Scrum-of-Scrums model. This massively distributed project was almost as productive as the small Scrum project with a co-located team. For a globally dispersed team, it is one of the most

[279] Stacey, R.D. 2001. Complex Responsive Processes in Organizations – Learning and knowledge creation.

[280] Kaufmann, S.A. 1993. The Origins of Order. Self-Organization and Selection in Evolution.

[281] Boisot, M. 1998. Knowledge Assets:203.

[282] Schwaber, K. 2001. Agile Processes and Self-Organization.

[283] Sutherland, J. Viktorov, A. Blount, J. 2006. Adaptive Engineering of Large Software Projects with Distributed/Outsourced Teams.

productive projects ever documented at a run rate of five times industry average. These results show that the Scrum methodology is scalable to large teams.

Nichols[284] did a similar study also claiming a team cluster size of around seven with an upper limit of twelve. Nichols illustrated a similar structure to Sutherland's Scrum-of-Scrums with team clusters interconnected via Team Leads and Role Managers. Teams of role managers not only make the network a small world, but also serve to make the network searchable, greatly shortening the average communications path. The results from Sutherlands Scrum-of-Scrums study are countering the mythical man-month theory by showing that it is possible to add more people to large complex projects and achieve surprisingly high productivity.

Boisot[285] lures companies to take advantage of technology that allows for delocalising teams without much loss of collaboration. The delocalisation and internationalisation of trust will enhance the ability to operate Social Learning Cycles in the lower regions of the I-Space independently of spatial and cultural constraints. It also allows development of systems round the clock.

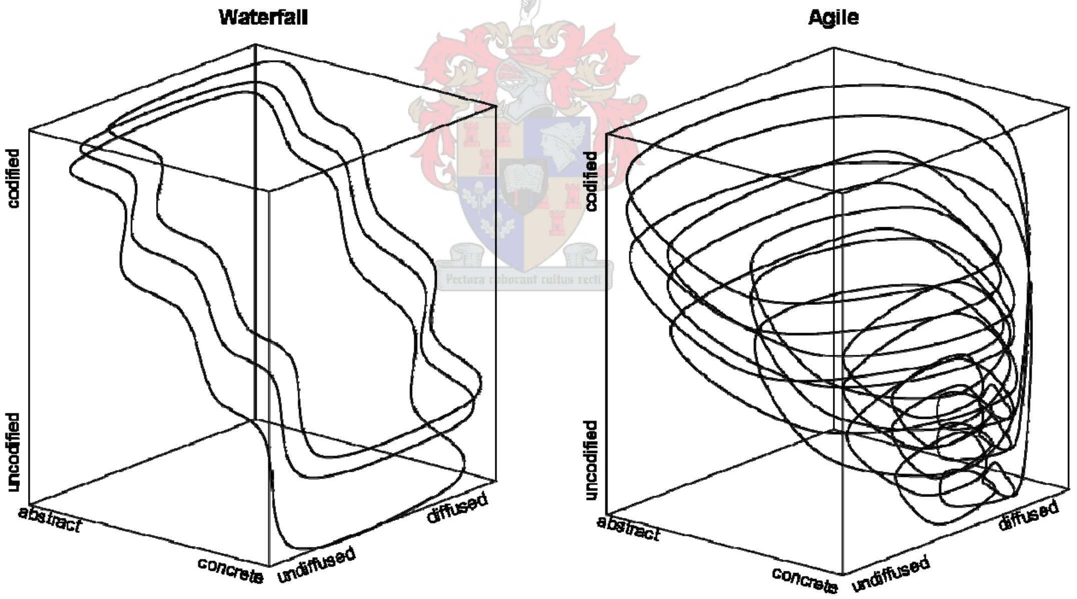


Figure 3-11: Social Learning Cycles for Waterfall and Agile SDLC models

The aim of the traditional waterfall and spiral lifecycles models was to follow a deep Scrumpeterian learning cycle as plotted in the I-Space above in Figure 3-11. However these

[284] Nichols, W. R. 2006. Building Successful Software Development Teams Using TSP and Effective Communication Networks.

[285] Boisot, M. 1998. Knowledge Assets:225.

cycles tended to progress too slowly and would over various product iterations gradually move upper and out of the E-max region. [286]

The I-Space depicted on the right is representative of an Agile lifecycle model. The movement shapes like a tornado with the eye anchored inside the E-Max region. It is a very fast Scrumpeterian cycle that harvests creative destruction and rapidly creates value through abstraction and codification towards V-max.[287] The Agile Development principles complements Boisot's six SLC phases very well:

- *Scanning.* Individuals and small groups identify threats and opportunities in available data and thereby turning such data into insights. Scanning may be very rapid when the data is well codified and abstract and very slow and random when the data is uncodified and context specific.
- *Problem Solving.* The process of giving structure and coherence to such insights.
- *Abstraction.* Generalising the application of newly codified insights to a wider range of situations. This involves reducing them to their most essential features and conceptualising them. Problem solving and abstraction work in tandem.
- *Diffusion.* Sharing the newly created insights with a target population. The diffusion of well codified and abstract data to a large population will be technically less problematic than that of data which is uncodified and context-specific.
- *Absorption.* Applying the new codified insights to different situations in a learning-by-doing fashion.
- *Impacting.* The embedding of abstract knowledge in concrete practices. The embedding can take place in artefacts, technical or organisational rules, or in behavioural practices. Absorption and impact work in tandem.

Boisot[288] defines teams as small groups drawn from the community and operating in a focused problem-solving mode in response to threats and opportunities. As teams progress through their life cycles and as both the problems they face and the solution they explore

[286] The E-Max region is in the bottom, right region of the I-Space cube where entropy is at its maximum. This is also known as the Chaotic Regime. The region is associated with high emergence and innovation.

[287] The V-Max region is in the opposite corner from E-Max and represents the maximum Value possible. This is also the area known as the Ordered Regime. Emergence and innovation is worthless if not codified, abstracted and first-to-market (early stage of diffusion). The need for being in both the E-Max and V-Max regions is what Boisot calls the Paradox of Value.

[288] Boisot, M. 1998. Knowledge Assets:228.

become better structured and understood, so team processes become more formalised and bureaucratic.

Agile teams require skills in integrating team and organisational processes in a seamless learning cycle. Agility can therefore be defined by capacity to harvest and transform entropy into economic value at a sustainable velocity within an innovative development ecosystem.

3.8 Sustaining an Innovative Development Ecosystem

Nonaka and Takeuchi's[289] five phases of organisational knowledge-creation as applied to system development practices could be defined as follows:

- *Sharing*. The team members' shares their mental models, perspectives, intuition, motivations and builds mutual trust and unity. The self-organizing team share their interpretations of their intentions. Management injects creative chaos by setting stretch goals and endows a high degree of autonomy. This is also a socialisation phase where the team gets to know one another to establish a *boundary-spanning unit*.
- *Creating concepts*. The shared mental model is formed when the team articulates it through continuous dialogue to formulate it into crystallised explicit concepts. The team autonomy allows for cooperative creative reflection, flux, chaos, and destruction. Rethinking their assumptions and converging their shared mental model through multiple reasoning methods such as deduction, induction and abduction.
- *Justifying concepts*. The team determines if the concepts are truly worthwhile and justified against the criteria determined collaboratively with top management.
- *Building archetypes*. The team creates a tangible model or prototype of the product concept as well as the blueprint. Attention to detail and dynamic cooperation of various departments is the key to managing this complex build process.
- *Cross-levelling of knowledge*. The team now needs to mobilise their creation to affiliated companies, customers, and other parties outside the company through dynamic interaction. Depending on the reaction or feedback, a new round of development is initiated.

Japanese car manufacturers have been overlapping these development stages in what is called the *rugby-style* to compress their new product introduction lead time. The approach also involved the production department from an early stage of the project, which leads to the

[289] Nonaka, I. Takeuchi, H. 1995. The Knowledge-Creating Company:85-89.

development of designs amenable to manufacturing. This also results in short production lead times and higher product quality. However since the approach depends on an interdepartmental pool of personnel who share the same space and time, the process is liable to give to much importance to preserving overall unity and conformance. In the European approach there is an intrinsic trade-off between stretched performance criteria and lead time, which is not the case in the Japanese approach where they manage to achieve both targets.[290]

Once the product concept is determined, all the functional departments move simultaneously, as in the rugby-style, running together to meet the targeted cost, performance level, and launch date. First, large-scale socialisation takes place, during which project members visit foreign markets to gain tacit knowledge. Second, and interdepartmental collaboration takes place to implement the overall business strategy, with departments sharing a common goal and a common information base. Third, all project members engage in evaluating or testing the prototype to judge whether the product concept has been realised.[291]

Many of the principles defined point to the creating and sharing of knowledge amongst all the team members. Nonaka and Takeuchi[292] defined five conditions for promoting the knowledge-creation. These conditions are essential in context of the establishment of a healthy Agile Development Ecosystem:

- *Intension*. Individual and collective commitment to the vision of what is to be achieved for the team and project to succeed.
- *Autonomy*. At the individual level all the members should be allowed to act autonomously as far as possible, thereby increasing the change of unexpected opportunities. Autonomy also increases the possibility that individuals will motivate themselves to be creative. It develops a holographic structure in which the whole and each individual share the same information. It is a system in which Morgan's minimum specification principle[293] is met as a pre-requisite for self-organising and autopoietic teams.
- *Fluctuation and creative chaos*. The team should foster an open and robust attitude to changes in the external environment that exploits ambiguity, redundancy and noise. Japanese companies often resort to the purposeful use of ambiguity and creative

[290] Nonaka, I. Takeuchi, H. 1995. The Knowledge-Creating Company:210.

[291] Nonaka, I. Takeuchi, H. 1995. The Knowledge-Creating Company:212.

[292] Nonaka, I. Takeuchi, H. 1995. The Knowledge-Creating Company:76-82.

[293] Morgan, G. 1997. Images of Organization:102.

chaos. Top management often employs ambiguous visions or so-called strategic ambiguity and intentionally creates flux with the team setting. Reflection-in-action is a required attribute to create order from chaos.

- *Redundancy*. This rugby-style redundancy that develop different approaches to the same project and then argue over the advantages and disadvantages of their proposals. Internal competition encourages the team to consider a variety of perspectives and with the guidance of a leader the team eventually develops a common understanding of the best approach. This style is also evident in natural reproductive systems where only the fittest survive.
- *Requisite variety*. According to Ashby,[294] the internal diversity must match the variety and complexity of the environment. Members can cope with many contingencies if they possess requisite variety, which is enhanced by combining information differently, flexibly, and quickly, and by providing equal access to information throughout the organisation.

Japanese brainstorming camps[295] are informal meetings for detailed discussions to solve difficult problems in development projects. The meetings are not limited to project team members but are open to any employees who are interested in the development project under way. In these discussions, the qualifications or status of the parties are never questioned, but there is one taboo: criticism without constructive suggestions. These camps are a medium for sharing experience and enhancing mutual trust amongst participants. It re-orientates the mental models of all individuals in the same direction, but not in a forceful way.

According to Larsen and Pixton[296] to lead an organisation through the change, senior leaders need a collaborative leadership style that encompasses the Agile principles. A collaborative leader convenes the right people and creates an environment of openness and trust. The leader lets team members decide on what to do and by when, and then steps aside and lets the team perform and produce. Begin with the right team members. Ensure that team members bring the necessary talent, the communication skills to interact on the team, and the ability to work interdependently. All team members must be open to collaboration. Build trust by

[294] Ashby, W.R. 1958. Requisite Variety and Implications for Control of Complex Systems. Cybernetica, Vol.1:83-99.

[295] Nonaka, I. Takeuchi, H. 1995. The Knowledge-Creating Company:63. The Japanese term for brainstroming camps is '*tama dashi kai'*.

[296] Larsen, D. Pixton, P. 2006. Team Collaboration for Senior Leadership. Agile Project Management Advisory Service.

believing team members will bring their best talents to the team efforts and work for success for all team members and themselves. Let the team decide what to do and by when. Working together, teams define goals, strategies, and measures of success, and determine by themselves how to hold each other accountable. Maintain an open environment that encourages the unencumbered flow of ideas, interactions, and discussions. Bring challenges and difficult topics to the table. Ensure a trusting environment that is non-judgemental. Work together to find solutions going forward, rather than looking back to place blame. Pause at the end of iterations or milestones to allow the team to incorporate learning and understanding, before re-evaluating goals and objectives. Create a place where people want to work.

3.9 Maximising Value Velocity

Christensen[297] derived a model for tracing the evolution of technology innovation that he names the technology S-Curve. When plotting a single innovation the curve takes on an S shape. The foot of the curve is the incubation period that consumes resources for little or no noticeable outcome. As an innovative disruptive technology or product concept emerges it suddenly shoots up in value. As the technology is diffused and incorporated into the mainstream of society its incremental innovative value flattens out as it fossilises over time.

To sustain maximum innovation is not easy. It is difficult to predict when emergence is due. It is difficult to plan for it. It is however possible to analyse and measure it. This measurement is defined by the rate of adding value. The control parameter for maximising productivity is complex as it relates to the various conditions for establish a healthy development ecosystem. A model that supports this empirical management style is be the Complex Adaptive Systems approach.

[297] Christensen, C.M. 1992. Exploring the Limits of technology S-Curve.

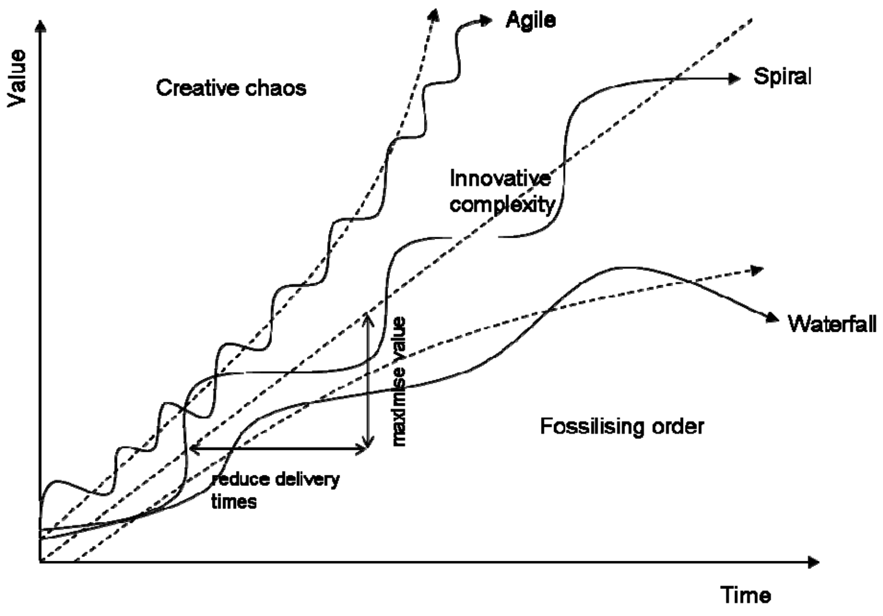


Figure 3-12: SDLC Technology S-Curves[298]

Christensen's S-Curve model provides another perspective on Boisot's Evolutionary Production Function and his Social Learning Cycle model. Figure 3-12 shows the three regions of complexity theory on the technology S-Curve as well as examples of the three popular SDLC models. The Waterfall model is a slow cycle that consumes the most resources. The Agile model has a higher frequency of adding bursts of value close to the edge-of-chaos. In between these two extremes one would find a more manageable even ground. The productivity measure can be derived as the mean angle of the curve by maximising value and reducing delivery times. This angle is the speed at which value is delivered.[299] It is clear that the Agile principles supports maximum velocity by increasing value and decreasing delivery time.

Christensen[300] discovered that principles such as better management, harder work and reduction in faults do not help to sustain innovation. The best management techniques have led their firms to failure. He argues that sustainable innovation depends on the emergence of disruptive technology breakthroughs. The conditions and rules that incubate these breakthroughs are very different to what is understood to be good management practices. Many other authors[301] concur.

[298] Adapted from the technology S-Curve. Christensen, C.M. 1992. Exploring the Limits of technology S-Curve.

[299] Value Velocity = Value Created / Delivery Delay.

[300] Christensen, C.M. 1992. Exploring the Limits of technology S-Curve.

[301] Sutton. 2002. Weird Ideas That Work – 11½ Practices for Promoting, Managing, and Sustaining Innovation. Davila, Epstein & Shelton. 2005. Making Innovation Work. Wind & Crook. 2005. The Power of Impossible

3.10 Conclusion

There exists a strong correlation between the principles of the SDLC methodologies that subscribe to the Agile Alliance and that of the new management approaches formulated around complexity theories and models, in particular the characteristics of Scrum and Complex Adaptive Systems.

Sketching the SLC curves of the Waterfall and Agile methodologies in I-Space provides a better understanding of the dynamic information flows and why Agile methods outperforms the Waterfall methods in velocity and value creation.

Plotting the technology innovation S-Curves of the Waterfall and Agile methodologies gives another perspective on these dynamics and shows how Agile methods yields frequent bursts of increasing value creation over reduced delivery times.

Thinking. Larsen & Pixton. 2006. Team Collaboration for Senior Leadership. Binney & Williams. 1995. Leaning into the Future.

Chapter 4
A case study of CI OmniBridge

Simple, clear purpose and principles give rise to complex, intelligent behaviour.
Complex rules and regulations give rise to simple, stupid behaviour.
Dee Hock, 1999

The purpose of the case study is to analyse the systems development life cycle methodologies employed by CI OmniBridge in order to measure a goodness-of-fit against the research described in the preceding chapters.

4.1 Introduction

CI OmniBridge is a global[302] company that develops information service delivery infrastructures that comprise of embedded computer hardware and firmware[303], multi-media communication middleware[304], and Internet service based information management software systems for fleet owners and fleet managers world-wide. Subsequent to a decade of exponential growth fuelled by exceptional product innovation and strategic market focus, the company has positioned itself as the potential echelon in its market segment. However, as is common in the information technology industry, the company's development department, the heart of its intellectual capital, is struggling to uphold its good reputation of delighting its various stakeholders with *satisfactory systems delivery*.

4.2 Sensemaking with Scrum

The original product development team was relatively small with a few embedded systems engineers focusing on hardware design (vehicle onboard computer platform), firmware design (embedded operating system and device drivers) and wireless communications middleware software design. The software development was done by another small team responsible for database design, backend server-side subsystems, desktop configuration and reporting software interfaces as well as web-based online interface software.

[302] The company has significant operations throughout Africa, Europe, Australasia and the Americas.

[303] *Firmware* is an industry term used to distinguish between desktop computer software and embedded device software, the firmware, as one would for example find running in mobile phones.

[304] *Middleware* is an industry term used to classify the software systems that coordinates and transports data communications between various distributed systems.

During one specific period, the project management attempts had not worked out well and the projects were running very late. The team was burning the midnight oil at least three nights a week with extensive system-wide testing, problem investigation and correction. Management did all they could to support the team. After thirteen test cycles the product was finally released to the market by a very exhausted team.

In their attempt to restore some vigour, the management team presented the next project proposal on which they spend a considerable effort in gathering detailed customer requirements. The development team however saw the requirements as completely unrealistic and unachievable scope within the given deadlines. The meeting was adjourned without any objectives or decisions taken. Both sides were *silently aroused*.[305]

Management then called a small task group together to resolve the tense climate. A new team leader was selected from the peer group and asked to spearhead the new project by forming a small team and driving the project forward. The user requirement was taken as input but dissected and prioritised into more realistically attainable work baselines. This process achieved immediate complexity reduction and restored sense and re-established anchors for the team to refocus on the work to be done.

The approach was to breakdown the unattainable scope into smaller, simpler components. Initial estimates were still difficult to determine but the team could at least predict effort by looking at each component in isolation. Critical milestones were pegged like planned stops along a journey. The interdependence of components and total effort determined what was possible by when. At that time it was the most practical approach to prioritise requirements. The main focus was to establish the core, critical, essential functionality. The team leader demonstrated personal commitment by taking on highest risk development component while entrusting the other team members to do the best possible work as a tightly knit team. Individuals outside of the project team were only subcontracted when it was absolutely necessary. The project was reasonably complicated and involved new hardware, firmware and software components to be developed. The following simple rules were defined:

- Each component is assigned to an *owner* who will be responsible for its on-time delivery.
- Break down the work into goals that can be achieved in a *weekly demonstrable* deliverable.

[305] Weick defines *autonomic arousal* as a sensemaking opportunity wherein the role of management is to reduce cues in order to restore organizational order.

- Conduct *daily meetings* of 5 minutes to check the alignment with the goal of reducing external interferences and focusing on achieving the goal.
- *Maintain a log* of issues that needs to be addressed during development.
- Work on *shrinking the logs*.
- Reduce ambiguity and ignorance – *coach* one another.
- Follow this lifecycle:
    - Envision (ideation);
    - Involve (knowledge sharing);
    - Enable (get things done); and
    - Deliver (close out and move on).

The strategy was to move the team forward step-by-step and getting work done smartly and resolving detail on the fly. Various internal demonstrations to management ensured visibility of the progress being made and provided an opportunity for gathering critical feedback on the emerging design direction.

Risk management was achieved by making a simple list of possible issues, accessing its impact, assigning an owner, estimating the probability of occurrence, and formulating a strategy to avert the risk.

A very important milestone was met on time at an international symposium. The final project was delivered slightly overdue. The productivity was the best ever experienced before and the project became a landmark success story in the department's history.

This approach to project management and systems development was borne in Scrum and XP principles. The development team itself has organised itself into various Scrum teams and a Scrum-of-Scrums meeting is held weekly. The approach was permanently adopted and incorporated into the Quality Management System based on ISO standards.

4.3 Quality Management System

CI OmniBridge formally defined its processes and values to establish a Quality Management System (QMS) based on the ISO 9001:2000 which it maintains through regular internal and external audits. The QMS has proven to be very effective by enforcing a discipline for recording decisions and metrics, as well as for general record keeping and configuration management.

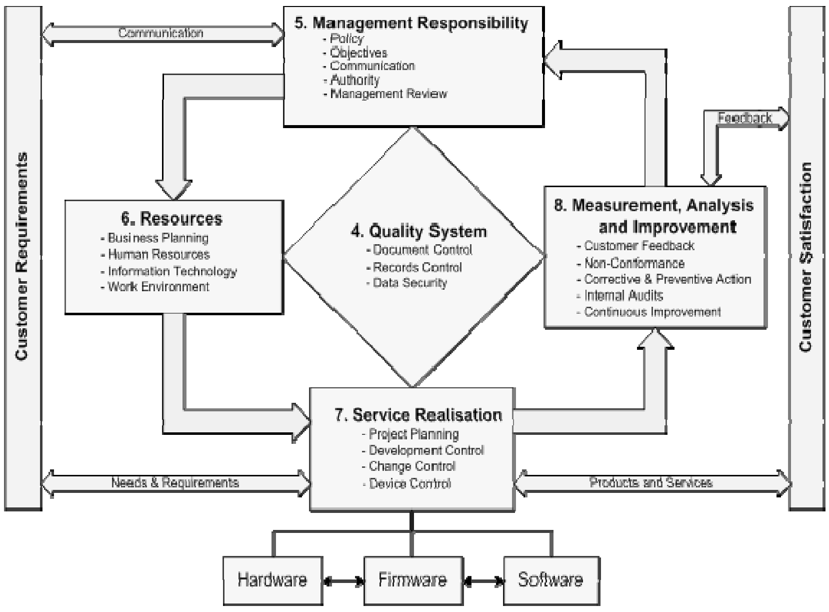


Figure 4-1: QMS Overview[306]

Figure 4-1 gives a high-level overview of the various process areas as defined in accordance to the ISO 9001 numbering scheme. Systems development processes primarily falls under the Service Realisation area and is concerned with the design and development of integrated hardware, firmware and software products.

Hardware, firmware and software are seen as three essential aspects of the company's Final Valuable Products (FVP). The SDLC embodied by the QMS is called Flows as it is drawn up as classical process flow charts containing processes with inputs and outputs, and decision points for redirection and guidance. Since hardware design and development contains the most process blocks and decision gates it is the predominant development flow from which the firmware and software branches out and meets during testing and integration phases.

The QMS primarily focuses on what processes and artefacts exists, but it does not focus on how these processes are to be performed. It provides standard templates for the artefacts with prompts for process guidance. The key business processes are broken into steps; each with required and recommended inputs and outputs. Decision points are recorded with review forms with applicable authorities and approvals as required by the QMS.

The Analysis Phase is conducted in conjunction with the customer, prior to starting a project and results in a decision regarding the start of a new project.

[306] Source: CI OmniBridge Quality Manual.

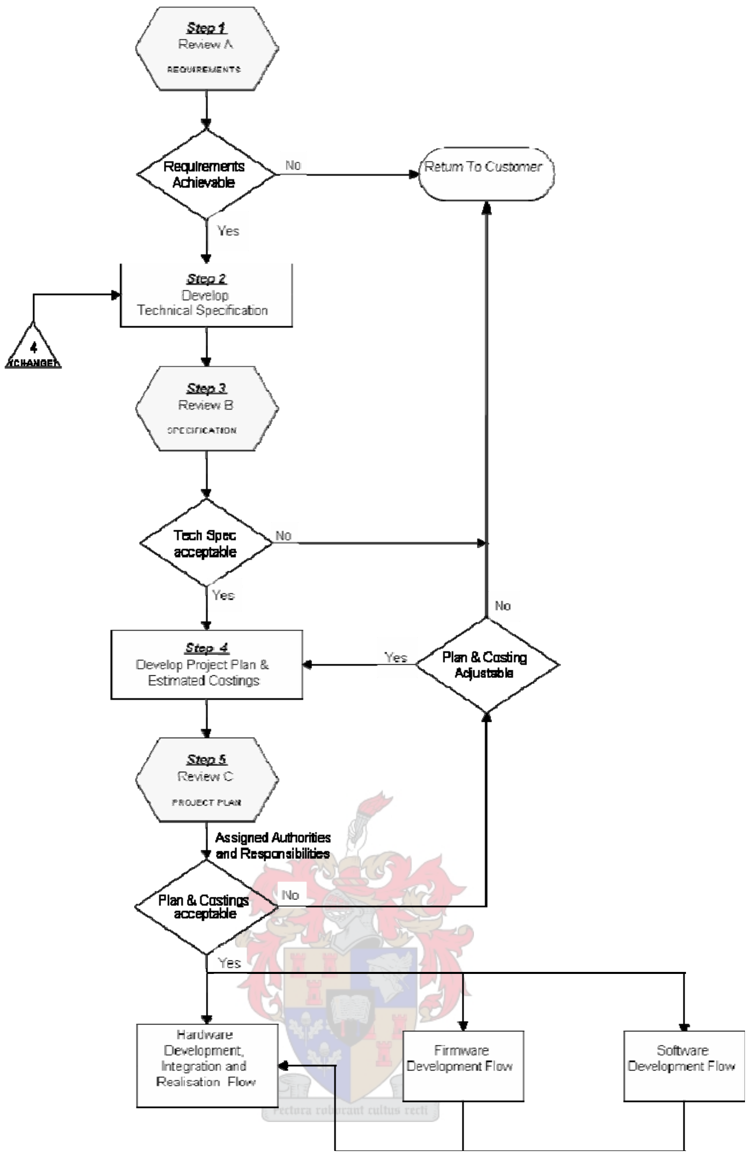

Step 1
Review A
REQUIREMENTS
Requirements Achievable
No
Return To Customer
Yes
Step 2
Develop Technical Specification
4
CHANGE
Step 3
Review B
SPECIFICATION
Tech Spec acceptable
No
Yes
No
Step 4
Develop Project Plan & Estimated Costings
Yes
Plan & Costing Adjustable
Step 5
Review C
PROJECT PLAN
Assigned Authorities and Responsibilities
Plan & Costings acceptable
No
Yes
Hardware Development, Integration and Realisation Flow
Firmware Development Flow
Software Development Flow

Figure 4-2: Analysis Phase Flow Diagram[307]

In accordance with Analysis Phase Flow, Figure 4-2, the project requirements are reviewed and the Technical Specification, Estimated Costing and Milestone Project Plan are developed. Project requirements are defined either, internally or externally by the Marketing Department on behalf of the customer. The requirements are reviewed and analysed to initiate a new project or amend an existing project. A Technical Specification is developed to meet the defined project requirements, and contains details of functionality to be developed. The customer representative, typically the product manager, approves the Technical Specification and Milestone Project Plan.

[307] Source: CI OmniBridge Quality Manual

Products are developed to meet the requirements of the Technical Specification. Development is conducted according to the relevant process flows. Any deviation from the agreed Technical Specification requires the customer's approval.

The Innovation Committee meets quarterly to review the Project Schedule. This team reviews progress of current projects against due dates and prioritises projects due to start, assigning desired start and end dates. Projects which have become current are assigned refined due dates based on current progress. Project teams are assigned to projects due to start, comprising developers from the three relevant disciplines. Team members are chosen on basis of current workload and specific skills or experiences required by project. A total of 42 business requirement concepts were presented at the Innovation Committee Meeting earlier in 2006. These concepts were categorized as follows:

- *Category A:* New business need – unlocks new untapped markets.
- *Category B:* Major enhancement – extends existing business need.
- *Category C:* Minor enhancement – enhancements to existing features.

Some of these concepts could be rationalized and normalized, albeit all are verified as real business needs that need to be delivered in about six weeks to six months. The category C concepts are fine-grain enough to be time-boxes. Categories A and B concepts would however require further in-depth analysis and breakdown.

During the Analysis Phase of each project, a Milestone Project Plan is developed. The project plan is split into two sections. The first indicates key development components that need to meet the requirements of the Technical Specification and estimated duration of work for each component. The second section shows deliverable milestones and target dates. Where a project consists of sub-projects, separate Milestone Project Plans are created for the parent project and for each sub-project. The Milestone Project Plan for the parent project shows the associated sub-projects. The completions of sub-projects are shown, as milestones, on the parent project plan. Project plans are reviewed at all reviews held during the development process to track progress and ensure due dates are met. If delivery dates are compromised, the customer is notified before resource cross-levelling and the planning is revised. Following each review, or at least weekly, the project leader updates the Milestone Project Plan, recording start and completion dates for all components, as well as any significant comments. These comments may indicate delays, early delivery and amendments to project. The Milestone section is also updated to show when a milestone was achieved and any significant comments regarding the status of the deliverables. If a review results in backtracking to an earlier review in the process flow, the milestones must be amended accordingly, indicating what additional reviews are required. The revised target date will then be recorded for all remaining milestones.

The Hardware Development Flow is the most elaborated flow with 28 steps, followed by the 8 steps of the software flow and 5 steps of the firmware flow. Apart from the three reviews A, B, and C defined in the Analysis Phase, the Design and Development Flows adds another 14 steps labelled from D to Q. Each step is iterative and the flow should not continue until all conditions are met at the review meeting. The steps are both necessary and straight forward. Typically a detailed design is followed by developing test records, unit testing, integration, integration testing, system testing and field trials.

One of the primary benefits of the QMS is that it makes performance quantifiable and transparent, albeit at a high-level and mostly based on subjective measures. The chart in Figure 4-3 illustrates non-conformance statistics over a particular period of review and clearly shows areas in need of improvement.

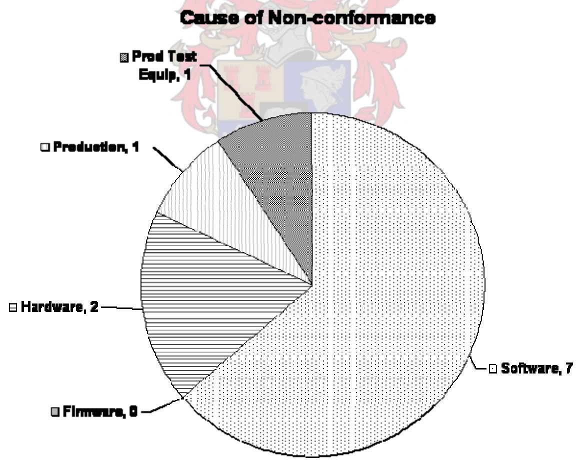


Figure 4-3: Non-conformance chart

In general more software projects results in non-conformances (defects) as well as being released very late and not to the satisfaction of the customer, as illustrated for a typical software project in Figure 4-4.

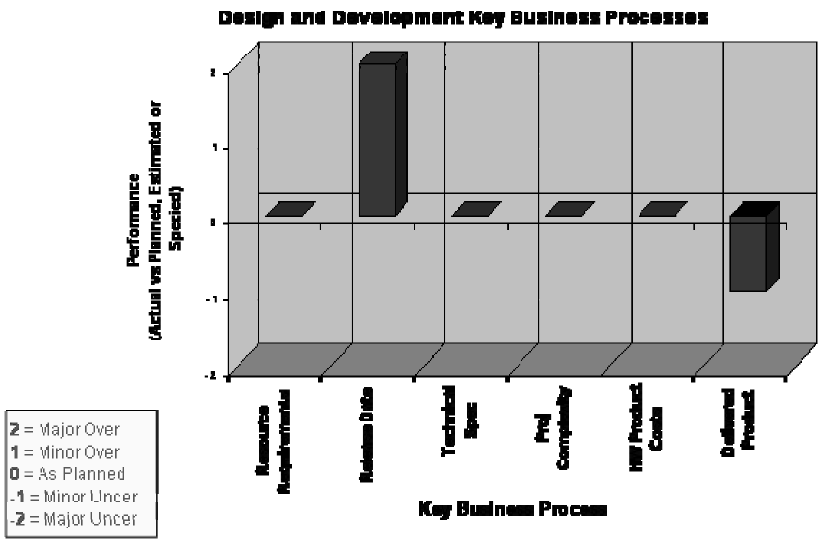


Figure 4-4: Key Business Process Performance Chart

The development capability capacity has not increased to reach the growing demand of customer requirements. Change in the business model was not managed well enough to envolve and grow the development department with the business growth. Informal silos form keeping the teams from knowledge sharing and serving the whole business. In such an environment even top achievers could lose their anchoring and slide into a reactive production mode.

4.4 Complex Adaptive Systems approach

The QMS provided mechanisms to maintain multiple current projects. The resource allocation and tasks of each project was however not aggregated onto a single perspective for driving resource utilisation and task prioritisation. Dee Hock's *Dirty Coffee Cup System*[308] was adopted to provide a simple solution to the problem on an experimental basis. A large white board was marked with vertical lines indicating calendar weeks and horizontal lines for each of the numerous current projects. Each project had an owner who was made responsible to identifying the project tasks and allocating resources to it. Each task was written on a small coloured square paper and stuck on the board. A piece of string with two magnets on each end was vertically positioned on the board to indicate the current date. As tasks are completed, the pieces of paper is taken off and put in a small paper bag. Tasks falling behind were clearly visible to the left of the date line. Ideally as people finish tasks and get freed up they should work on ad-hoc tasks that has fallen behind, before working on future tasks.

The system swiftly became chaotic as many of the projects started to fall too far behind and people did not even have the time to maintain their planning tasks.

[308] Hock, D. 1999. Birth of the Chaordic Age. Berret-Koehler.

According to Hock and his team at VISA, they had enormous success. The difference was that they had critical mass and only one project with many tasks. Hock notes that the few people who could not adjust to the diversity, complexity and uncertainty of the processes, were replaced with dozens of new people.

Complex Adaptive Systems require that most of the requirements for sustainability are met. It would have been more successful with more people and fewer projects; and with more focus and less peripheral distractions.

The project dashboard shown in Figure 4-5 provides a highly visual impact of all the critical factors of projects that are underway. The respective Scrum burn-down charts are also shown.

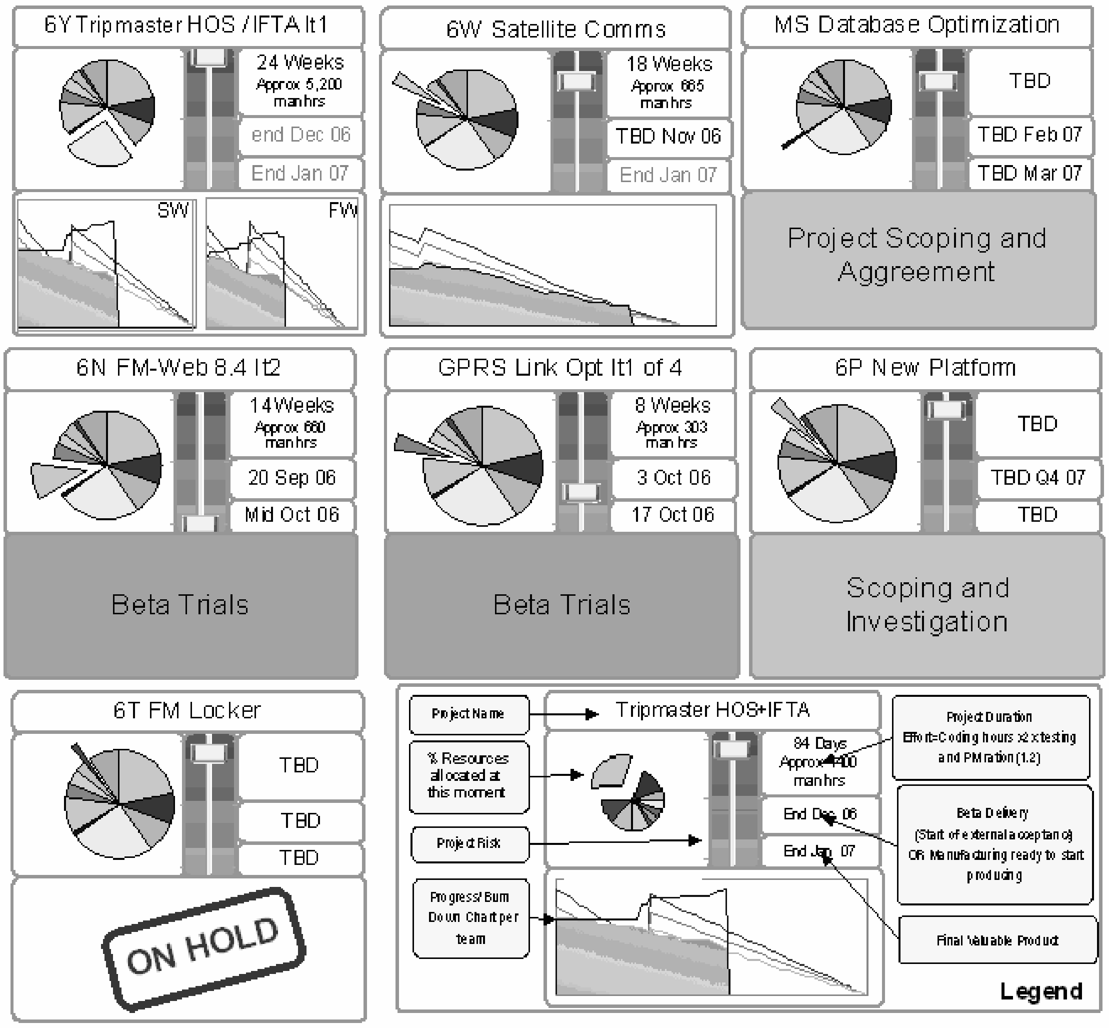


Figure 4-5: Project Dashboard

4.5 Survey

A short survey identified some common barriers, needs and comforts amongst the developers. The questions were similar to those asked during the daily Scrum meetings:

- *Barriers*: What stops you most from getting your work done as you intended, that if taken away would make you happy?
- *Needs*: What would help you most to get stuff done and find bigger challenges, that if given would make you happy?
- *Comforts*: What is currently your most valuable tool or help, which if taken away would make you angry?

The survey questionnaire gave examples to guide the thinking process; and it was intended to be stimulating and fun. The summary of the results are:

- *Barriers:* Unsolicited distractions. Lack of communication on perceived progress and management expectations. Long meetings. Lengthy documentation. High level of non-work related distractions in office. Excessive paperwork. Being micromanaged. Stress. Inappropriate e-mails. Meetings without agendas. Micromanaging others. Crisis management.
- *Needs:* More developers. Better upfront modelling of the problems and ideas. Individual white boards. More successes that build confidence and trust, which results in more successes. Training on tools and processes. Testing tools. Detailed single document specifications and centralized access to it. Dual displays. More test equipment. More recognition for delivering quality output. Better time management. Allowing more time for refactoring. More time to focus on new design.
- *Comforts:* The Internet. Google. Verbal idea sharing. Dual displays. Flexitime. Test equipment and productivity tools.

The team is steadily gaining critical mass. The project management team is doing their best to define priorities between the primary stakeholders and scheduling the projects onto a Product Development Roadmap, while removing barriers, satisfying essential needs and maintaining privileged comforts of the team.

4.6 Values

CI OmniBridge has been, and still is, undergoing tremendous changes. The company started off as a product design, manufacture, marketing and support organisation. The business is now undergoing an ambitious strategic shift towards becoming the biggest, global information service provider for managing commercial fleets. Whereas its predominant staff complement of a few years ago was a few dozen engineers, it has now grown to a couple of hundred customer facing staff members with new roles such as operations managers, customer relationship managers, helpdesk operators, key account managers, regional sales managers, product managers, and fleet consultants.

For the company's pioneers and veteran staff, a strange new culture is emerging with emphasis on service delivery and customer satisfaction. This dissonance matches with the model Greiner defined for organisational evolution. It has established itself in the Fleet Management market through relatively small innovative steps and thereby earned a dominant market position.

More recently the company harvested the insights of Clive Howe[309] to align the whole company using his compass aligned performance system. CI OmniBridge is set out to become the leader in global information services for the management of commercial fleets by instilling the following values into the team:

- *Accountability* – being accountable for getting things done with a sense of urgency while taking the company's best interests into account.
- *Creativity* – encouraging innovation and listening to new and creative ideas and different ways of doing things whilst avoiding negative or dismissive behaviour.
- *Trustworthy* – display and encourage honesty, dependability and reliability in others.
- *Work smart* – always applying mindfulness to all tasks to ensure the desired outcome is achieved in an optimal manner.
- *Integrity* – being open and honest with each other and conduct all business in an ethical manner.
- *Service culture* – delivering excellent service in a customer focused way.

[309] Clive Howe, C. 2003. Simple solutions to strategic success – the one-page c@ps planning process. Knowledge Resources Publishing.

- *Entrepreneurial spirit* – seeking new commercial opportunities and being prepared to change to take advantage of them, without deviating from the core business.

The vision, values and critical success factors are written into its Quality Management System and promulgated throughout its various geographically dispersed locations around the globe. These values complement those of the Agile Development community.

4.7 Conclusion

CI OmniBridge has successfully adopted Extreme Programming and Scrum principles and practices. The Quality Management System conforms to ISO standards and although it does not specifically prescribe Agile methodologies, the processes were influenced by the prior successful experience based on XP and Scrum. The teams are motivated and driven by a set of shared values that complement the Agile Manifesto values. The Complex Adaptive Systems approach will only work if a comfortable critical mass is reached.

Chapter 5
Conclusion

Speaking and writing is an ever renewed struggle to be both apposite and intelligible, and every word that is finally uttered is a confession of our incapability to do better.
M. Polanyi, 1958

The preceding chapters are partially representative of extant research underway to find the evasive silver bullet. All of the many SDLC methodologies had in its time been effective in the context of its time and application.

For the last fifty years project managers had designed models of *how they think* systems development should work by imposing formal phases, rules and standards. The various stages of the lifecycle where bounded and isolated to different and disparate workgroups such as analysts, designers, coders, and testers. These groups were socially and geographically separated as much as they were functionally separated. Their primary means of interaction was through formal written documentation. In large corporations and government agencies, systems development projects failed more often that it succeeded. Stakeholders wanted answers. The frantic search for the silver bullet has not subsided since.

The Chaos Report does not make any sense. Instead it presents a list of external sources of blame such as drawing attention on the lack of executive support, user involvement and experienced project managers. They add little more awareness to what Brooks published three decades ago. Product development departments are instead suffering from Senge's Learning Disabilities.

Companies that made Information Technology their life-blood seemed to have found the secrets of success, constantly innovating and delivering final valuable software products. Several veteran and incumbent experts joined forced to discover how systems development actually works inside these successful *super-productive teams*.[310]

[310] Jeff Sutherland and Charles Schwaber reported average productivity differences in software teams to be 600%. The most successful team to date, Borland Quatro Project, reported to be 3000 times more productive. Source: http://jeffsutherland.com.

As the incumbent Information Age is being interwoven with the more mature Industrial Age both are being interwoven with the ancient Biological Age. The proliferation of wireless interconnected devices and accelerated progress in nanotechnology is made possible by these interwoven threads of technological innovation.

The system development lifecycle is a complex process of incremental iterative moments of creativity. Each moment determines the path of for future moments. The attributes of good systems development lifecycle management are dominated by that of sensible leadership. It is a healthy, cognitive process, acting out as it would during a rugby game. A strategy is executed, tested and adapted, moving the ball vigorously forward.

The last decade has seen an increasing interest in knowledge management as a consequence of the rapid development of information systems and technologies, which enable both private and public sector organisations to leverage their knowledge assets far more effectively than was hitherto possible. Given the rapid emergence of cost-effective Internet-powered technologies, it is now possible for global enterprises to effortlessly and instantaneously communicate and share information across their geographical and functional structures. This collaboration across organisational boundaries has become a critical success factor and source of competitive advantage.

Information Technology has helped managers with the augmentation, classification, filtering, visualisation, extrapolation, analysis, and forecasting of data. As the Information Age is being woven into the Industrial Age, organisations start shifting its measure of value away from the physical world toward the meta-physical world. Thinking of teams and organisations as living Complex Adaptive Systems essentially acknowledges the past mistakes of treating it as closed mechanical systems.

Apart from briefly citing Nonaka, Takeuchi, Senge, and Argyris there is little in-depth study of new management thinking being incorporated into SDLC methodologies. However, what emerged from the search for models and principles is a potential explanation and characterisation of the super-productive development teams as Complex Adaptive Systems. Conversely, acknowledging that the theory of Complex Adaptive Systems started in computer science, new management thinking at large has only recently shown interest in it and would probably adopt the lessons to be learned from future SDLC methodology research.

The systems design process relies on the *importance of drawing pictures as a way of seeing.* One should literally ask oneself the following sensemaking question: *How can you know what you need to build if you can not see what you are designing?* This is especially true with difficult, complicated, and cognitive dissonant problems which are common in the Information Technology industry.

As with Shackleton's expedition team, systems development often calls for the use of multidisciplinary teams. This team must include representatives from the various scientific, engineering, and business specialties, such as electronic and mechanical design, software engineering, manufacturing, reliability engineering, maintainability, logistic support, life cycle cost analysis, human factors, quality assurance, marketing, and management. All of these disciplines have unique views of product and process design that are constructed from their individual past experiences. These views are expressed in different ways that often use specialised notations and languages. These must however be shared by all of the team members. This requires an increase in the communication of product and process information among the members of the multidisciplinary teams. Without the constant flow of ideas, information, and analysis data, system development will not be effective. For the team members to be able to share other points of view, scientific collaboration requires communication beyond that of information alone to provide complete sensory integration.

Establishing a large web or interconnected nodes does not ensure that the system can handle high volumes of requests. In the same sense that the human brain is constructed it can only handle a steady rate of up to a dozen impulses per second. Creativity is however only possible at much lower clock speeds. The brain quickly becomes inefficient when overloaded. It is critical to maintain a steady drumbeat and establish an optimal rhythm for maximum productivity. However, in the context of Einstein's Brownian movement theory, particle-count also does not matter. Collaboration occurs in temporal moments of contact between two particles or, in terms of the Complex Adaptive Systems model, between two agents. Seldom more than two agents would be instantly involved in a singular collaborative moment.

The answer is not combating detailed complexity but instead the management of dynamic complexity. This is done through simple parameters such as Kaufmann's interconnection control parameter and through simple yet effective performance indicators such as the burn-down charts and sprint velocity as proposed by the Agile Scrum approach.

Another valuable insight made possible by the decreasing costs of computing and communication capacity is that systems developers reap the benefit of rich multimedia artefacts in order to maintain velocity closer to the maximum entropy region. In less technical terms this implies for example that the real voice-of-the-customer could actually be recorded as opposed to lengthy written technical specifications. Technologies such as online text, voice and video chatting or blogging enhances the customer-on-site experience. Digital still and video cameras can capture instant customer details. Conversely developers could capture and release sneak previews of the system-under-development to delight the customer with progress without having to endure the costs of supporting Beta[311] software installations.

Each bird in a flock knows how to fly and is fit to fly. It also knows the rules of behaviour and how to communicate effectively while in flight. These lessons are taught in the nests and during playtime as juveniles. Once the flock is airborne it knows how to act and there is little time for study. It maximised on every opportunity to achieve its goal with least dispersed energy. The flock however does not reach its destination in a single flight. It needs to take rests and recharge on potential energy levels.

The development of complex systems requires a certain incubation period in order for emergence to occur. The occurrence of emergence stimulates spontaneous eruptions of subsequent emergence in a *fractal tree chain reaction pattern*. For example, the first emergence entices two more occurrences, which in turn each triggers another pair of emergent products, and so forth. In open systems this entropic behaviour could be identified as creativity, but this could also lead to nothing or even destruction. It is necessary to measure emergence in order to direct it towards an end goal.

For example, when a developer is given a task to accomplish, it is expected that she would produce the best possible solution. However it is not possible to know what is possible until one sees the result of a first deliverable. This deliverable must however be useful and valuable albeit not final or perfect. This emergence is manifested in artefacts. Artefacts that can be interacted with are the only measure of production in such creative accomplishment.

The reason why traditional management finds knowledge management so difficult is because of this missing link between the mechanical manufacturing of tangible artefacts and the artful knowledge that creates other types of tacit value. Implicit knowledge works in tandem with its complementary explicit knowledge. There is an abundance of extant explicit knowledge in the world. Aspiring to gain a *complete* understanding is fatal. Mastering the knowledge through small learning cycles is more achievable, yielding efficiency, effectiveness and

[311] *Beta* is a label given to a specific prerelease version of a system that is ready for external testing and evaluation, but has not yet been fully qualified as the final product release. The primary purpose of the Beta release is to get early customer feedback before giving them the final solution only once the project is complete. As can be guessed Beta is preceded with an internal *Alpha* release. Using remoting technologies it would be possible to expose external stakeholders to Alpha releases as well.

satisfaction. Social networks such as newsgroups, blogs, forums and Communities of Practices, are only really valuable if the knowledge worker is an active contributing member. At least one member of the team working in a specific domain needs to reach this *switched-on* level of mastery.

Innovation *impacting* is more effective than the hard driven need for urgency. Avoid psychological disposition of ownership and control causing staff to feel like physical assets with little or no leverage and control over the destiny and success of the company. Domain knowledge grows during the development stages and is not known upfront. In many cases the important domain knowledge is only discovered once the first version is installed and being used. Focus on the skill of managing product creation *with* creative people opposed to telling them how and what to do, in a recipe like manner.

Make a clear distinction between manufacturing and crafting. Systems development is essentially an act of crafting. It is the *mindful*[312] creation of useful artefacts. An artefact may however be the end-to-end procedure for manufacturing of a product or a recipe for soup. The process of manufacturing of physical motor vehicles is much different to that of designing a new motor vehicle model. However the manufacturing needs to conform to strict discipline and standards to produce reliable quality vehicles as the designers intended it to be. As such the designers are the creators, not of cars, but of models, such as the original Ford *Model T*. A century of management science was invested into manufacturing processes. Software manufacturing on the other hand is relatively simple to accomplish as it is easily automated. Software products can be transported and replicated with much less effort and costs than any other kind of product. It obeys the laws of knowledge opposed to the laws of physics.

Software is seen by law as a copyrightable intellectual artefact alongside books, music and paintings. There exist many schools for acquiring these skills and learning about the technicalities of these art forms. They do not, however, prescribe a set of standard formulas or recipes for authoring a bestseller novel. At the very best students would gain some insights, principles and guidelines. Many of these artistic talents can only be developed from within.[313] For example, to learn to dance or fiddle requires one to observe what other scholars have mastered and then to reflect upon ones own enactments. In software design

[312] Weick, Snowden and Stacey use the term 'mindful' to represent a holistic attentiveness in the process of Sensemaking in organisations.

[313] Satori - the Zen Buddhist term for enlightenment means '*to understand*'.

this is achieved by pair-programming and should not be confused with peer-programming. Pair-programming ideally requires an experienced developer alongside a junior developer, not two equally competent peers.

Make time for that what needs to occur naturally. As breathing is essential to performing ones work; so are many other social distractions. Accept this and build it into the SDLC. For example, allot one hour per day for playful distraction. The benefits are important in terms of stress relief, creating a playful and fun loving ecosystem and most important to stimulate creativity.

Reliable delivery of final valuable products is a realistic business objective that is too often compromised by overcomplicated rigid processes. Reliable systems delivery can however be assured by applying a strategic management methodology that is focused on adroit autonomous team innovation and an integrative body-of-knowledge framework that is influenced by the new Knowledge Management theories such as Complex Adaptive Systems theory.

Change prevails and conquers society and technology. Abstract meta-models have longer lifetimes since it is not restricted to a specific instance and can be tailored to fit almost any requirement. Meta-models are however not very practical otherwise. The most appropriate model is that of Complex Adaptive Systems which only requires a certain critical mass and a practical set of rules for guidance. In Complex Adaptive Systems, the goal is not reaching the planned destination, but instead making the most of the moments towards getting there whilst guided by a clear vision.

The thesis traversed full circle from modelling lifecycles and returned to the realisation that it is living systems that is in need of nourishment. The recent attention given to biotechnology may once again open the eyes of practitioners and researchers to embark on a scientific expedition to discover the secrets of how life itself has endured its inherent complexities and resulted into such profound beauty.

Bibliography

ALHIR, S. 2002. Understanding the Unified Process. *Methods & Tools Newsletter*. March.

ALLEN, D. 2002. *Getting things done – the art of stress-free productivity*. Penguin Books.

AMBLER, S.W. 2006. Survey Says Agile Works in Practice. *Dr. Dobb's Journal*. 3 August.

AMBLER, S.W. 2005. A Manager's Introduction to The Rational Unified Process (RUP). <*www.ambysoft.com/downloads/managersIntroToRUP.pdf*>. Accessed 30/10/06.

AMBLER, S.W. 2006. Choose the right software method for the job. <*http://www.agiledata.org/essays/differentStrategies.html*>

ANDERSON, D.J. 2006. CMMI DOI Comparison. <*http://www.apln.org/cmmi.html*>. Accessed 30/10/06.

ANDERSON, D.J. 2005. Stretching Agile to fit CMMI Level 3 - the story of creating MSF for CMMI Process Improvement at Microsoft Corporation. *Agile Conference*.

ARBINGER INSTITUTE. 2002. *Leadership and Self Deception – Getting Out of the Box.* Berrett-Koehler.

ASHBY, W.R. 1958. Requisite Variety and Implications for Control of Complex Systems. *Cybernetica*, Vol.1:83-99.

BACH, J. 1995. The Challenge of Good Enough Software. *American Programmer Magazine.*

BARESI, L. DI NITTO, E. GHEZZI, C. 2006. Toward Open-World Software: Issues and Challenges. IEEE Computer. Vol.39. No.10.

BARLEY, S.R. KUNDA, G. 2004. *Gurus, Hired Guns, and Warm Bodies – Itenerant Experts in a Knowledge Economy*. Princeton University Press.

BASKERVILLE, R.L. MYERS, M.D. 2002. IS as a Reference Discipline. *MIS Quarterly* Vol.26 No.1.

BATESON, G. 1973. *Steps to an Ecology of Mind*. Granada Publishing.

BAXTER, A. 2005. Rapid results without a rugby scrum. *Financial Times London.* 27 July.

BECK, K. 1999. *Extreme Programming Explained: Embrace Change*. Addison-Wesley.

BECK, K. FOWLER, M. 2002. *Planning Extreme Programming*. Addison-Wesley.

BEREIT, M. 2006. Escape the software development paradigm trap. *Dr. Dobb's Journal*. 29 May.

BERKUN, S. 2005. *The Art of Project Management*. O'Reilly.

BINNEY, G. WILLIAMS, C. 1996. *Leaning into the Future – Changing the Way People Change Organizations*. Nicholas Brealey.

BLEICHER, P. 2003. Software Life Cycles and Kitchen Sinks. *Applied Clinical Trials Magazine*. October.

BOEHM, B.W. 1979. Software Engineering - As it is. *Proceedings of the 4th International Conference on Software Engineering*. IEEE Press.

BOEHM, B.W. 2000. Spiral Development: Experience, Principles, and Refinements. *Spiral Development Workshop*. CMU/SEI-2000-SR-008.

BOISOT, M. 1998. *Knowledge Assets – Securing Competitive Advantage in the Information Economy*. Oxford.

BOISOT, M. CANALS, A. 2004. Data, information and knowledge: have we got it right? Online Working Paper. *IN3: UOC*. (Working Paper Series; DP04-002) <*http://www.uoc.edu/in3/dt/20388/index.html*>. Accessed 30/10/06.

BROOKS, F.P. 1975. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley.

BROOKS, F.P. 1987. No Silver Bullet - Essence and Accidents of Software Engineering *Computer*. Vol.20 No.4.

BRYSON, B. 2003. *A Short History of Nearly Everything*. Doubleday.

BUSH, V. 1945. As We May Think. *The Atlantic Monthly*.

CASTANEDA, C. 1971. *A Separate Reality*. Penguin Books.

CASTELLS, M. 2000. *The Rise of the Network Soceity*. Blackwell Publishing.

CHRISTENSEN, C.M. 1992. Exploring the Limits of technology S-Curve. *Production and Operations Management Journal*.

CHRISTENSEN, C.M. 1997. *The Innovator's Dilemma – The revolutionary book that will change the way you do business*. HarperBusiness Essentials.

COAD, P. YOURDON, E. 1990. *Object-Oriented Analysis*. Prentice Hall

COAD, P. YOURDON, E. 1991. *Object-Oriented Design*. Prentice Hall

COCKBURN, A. 2003. People and Methodologies in Software Development. PhD dissertation. *University of Oslo*.

COLEMAN, H.J. Jr. 1999. What Enables Self-Organizing Behavior in Businesses. *Emergence*. Vol.1.Issue.1.

COOPER, M. 2001. Everyone is wrong. *Technology Review*. June.

COX, B. 1995. No Silver Bullet Revisited. *American Programmer Journal*. November.

CROSBY, P.B. 1979. *Quality is Free*. McGraw-Hill.

DAVILA, T. EPSTEIN, M. J. SHELTON, R. 2006. *Making Innovation Work, How to manage it, measure it and profit from it*. Wharton School Publishing.

DAVIS, A.M. 1992. Operational Prototyping: A New Development Approach. *IEEE Software*. Vol.9. No.5.

DAVIS, A.M. 1995. *201 Principles of Software Development*. McCraw-Hill.

DIJKSTRA, E.W. 1968. GOTO Statement Considered Harmful. *Communications of the ACM*. Vol. 11(3):147-148.

DOVE, R. 2006. Engineering Agile Systems: Creative-Guidance Frameworks for Requirements and Design. *4th Annual Conference on Systems Engineering Research (CSER).*

DRUCKER, P. 1988. The Coming of the New Organization. *Harvard Business Review*. Vol.66 Iss.1

DRUMMOND, H. 2001. *The Art of Decision Making – mirrors, of imagination, masks of fate*. Wiley.

EELES, P. 2005. RUP for Successful J2EE Projects. Rational Software. *IBM Software Group.*

EVANS, I. 2006. Agile Delivery at British Telecom. *Methods & Tools*. Summer.

FIRESMITH, D. 2006. OPEN Process Framework. <*http://www.opfro.org/Overview/Metamodel.html*>. Accessed 30/10/06.

FISHMAN, C. 2000. The Total Teamwork Agenda. *F@st Company*. Issue 33.

FLOOD, R.L. 1999. *Rethinking the Fifth Discipline*. Routledge.

FLOOD, R.L. JACKSON, M. 1991. *Creative Problem Solving*. Wiley.

FOWLER, M. HIGHSMITH, J. 2001. The Agile Manifesto. *Dr. Dobb's Journal*. July.

FRAME, J.D. 2002. *The New Project Management*. Jossey-Bass.

GEORGE, M. ROWLANDS, D. KASTLE, B. 2004. *What is Lean Six Sigma?* McGraw-Hill.

GIARRATANO, J. RILEY, G. 1989. *Expert Systems – Principles and Programming*. PWS-Kent.

GILLISPIE, C.C. 1960. *The Edge of Objectivity*. Princeton Paperbacks.

GLASS, R.L. VESSEY, I. 1998. Focusing on the Application Domain – Everyone Agrees It's Vital, But Who's Doing Anything About It? *IEEE Computer*.

GLASS, R.L. 2003. *Facts and Fallacies of Software Engineering*. Addison-Wesley.

GLASS, R.L. 2004. Matching Methodology to Problem Domain. *Communications of the ACM*. Vol.47. No.5.

GLOGER, B. 2006. Comparison of Methodologies. <*http://www.scrumalliance.org/index.php/content/download/6306/65109/file/ComparisonofMethodologies.pdf*>. Accessed 30/10/06.

GOLDSTEIN, J. 1999. Emergence as a Construct: History and Issues. *Emergence*. Vol.1. Issue.1.

GREMBA, J. MYERS, C. 1997. The IDEAL Model: A Practical Guide for Improvement. Software Engineering Institute (SEI) publication. *Bridge*. Issue 3.

HAMEL, S. HIGHSMITH, J. 2000. Optimize - or Adapt. *Software Development*. April.

HAMMER, M. 1996. *Beyond Reengineering – How the process-centered organization is changing our work and our lives*. HarperBusiness.

HEYLIGHEN, F. JOSLYN, C. 2001. Principia Cybernetica Web (Principia Cybernetica, Brussels). <*http://pespmc1.vub.ac.be/reqvar.html*>. Accessed 30/10/06.

HIGHSMITH, J. 1997. Messy, Exciting, and Anxiety-ridden: Adaptive Software Development. *American Programmer*. Vol.10.No.1.

HIGHSMITH, J. 2002. What Is Agile Software Development? *CrossTalk. The Journal of Defense Software Engineering*. October.

HIGHSMITH, J. COCKBURN, A. 2001. Agile Software Development: The Business of Innovation. *IEEE Computer*.

HOCK, D. 1999. *Birth of the Chaordic Age*. Berret-Koehler.

HOCK, D. 2000. Back to Nature. *CIO Magazine*. 15 January.

HOFSTADTER, D.R. 1979. *Gödel, Escher, Bach – An eternal golden braid*. Vintage Books: New York.

HOWE, C. 2003. *Simple solutions to strategic success – the one-page c@ps planning process*. Knowledge Resources Publishing

IMMELMAN, R. 2003. *Great Boss Dead Boss – How to exact the very best performance from your company and not get crucified in the process.* Phaice.

ISO. 2000. *SABS ISO 9000:2000 edition 2*. SABS. ISBN 0-626-12810-2

ISO/IEC. 1994. *Information technology -- Open Systems Interconnection -- Basic Reference Model: The Basic Model.* International Standard 7498-1.

JACKSON, M. C. 2003. *Systems Thinking – Creative Holism for Managers*. Wiley.

JACOBSON, I. 1992. *Object-Oriented Software Engineering*. Addison-Wesley.

JACOBSON, I. WEI NG, P. SPENCE, I. 2006. The Essential Unified Process. *Dr. Dobb's Journal*. 2 August.

JENSEN, W.D. 2000. *Simplicity – The New Competitive Advantage in a World of More, Better, Faster*. Perseus Publishing.

JOHNSON, G. 2001. All Science is Computer Science. *New York Times*. March 25.

JONES, C. 2006. Social and Technical Reasons for Software Project Failures. *CrossTalk - The Journal of Defense Software Engineering.* June.

KAST, F. E., ROSENZWEIG, J. E. 1970. *Organization and Management – A systems approach*. McGraw-Hill.

KAST, F. E., ROSENZWEIG, J. E. 1979. *Organization and Management 3rd Edition – A systems and contingency approach*. McGraw-Hill.

KAUFMANN, S.A. 1993. *The Origins of Order - Self-Organization and Selection in Evolution.* Oxford University Press.

KAY, R. 2005. CMMI. *Computerworld*. 24 January.

KREBS, J. 2005. RUP in the dialogue with Scrum. *IBM Rational Edge*.

KRUCHTEN, P. 2005. Software Design in a Postmodern Era. *IEEE Software*.

KURTZ, C. F. SNOWDEN, D. J. 2003. The new dynamics of strategy: Sense-making in a complex and complicated world. *IBM Systems Journal*. Vol. 42. No 3.

LARMAN, C. 2004. *Agile & Iterative Development – A Manager's Guide*. Addison-Wesley.

LARSEN, D. PIXTON, P. 2006. Team Collaboration for Senior Leadership. *Agile Project Management Advisory Service Executive Update.* Vol. 7, No. 6

LEEDOM, D. K. 2001. Sensemaking Symposium Final Report. *Command and Control Research Program Office of the Assistant Secretary of Defense for Command, Control, Communications and Intelligence.*

LIKER, J.K. 2003. *The Toyota Way - 14 Management Principles From The World's Greatest Manufacturer*. McGraw-Hill.

LINDER, G. 2006. Evaluation of Software Development Projects using the Agile Method Scrum. Masters Thesis. Software Engineering Research Group. *Lund University.*

MACKAY, H. 2005. Mind your own busyness. *The Age*. 17 September.

MACKENZIE, D. 2000. A view from the Sonnenbichl: on the historical sociology of software and system dependability. *Proceedings of the International Conference on History of Computing: Software Issues*. Springer-Verlag. New York.

MAGEE, S. 2002. ISO/IEC 15288 The System Life Cycle Process standard for the 21st century. <*http://syseng.omg.org/_ISOIEC15288.pdf*>. Accessed 30/10/06.

MAR, K. SCHWABER, K. 2002. Scrum with XP. Book chapter in Agile Software Development with Scrum. *Informit*.

MARKOFF, J. 2005. *What the Dormouse Said – How the 60s Counterculture Shaped the Personal Computer Industry*. Viking.

MARTIN, J. 1991. *Rapid Application Development.* Macmillan Publishing.

MEYER, B. 1988. *Object-Oriented Software Construction*. Prentice Hall.

MEYER, C. DAVIS. S. 2003. *It's Alive – The coming convergence of information, biology and business*. Crown Business.

MINTZBERG, H. 1987. The Strategy Concept II: Another Look at Why Organizations Need Strategies. *California Management Review*. Vol 30. No 1.

MOORE, J. 2006. ISO 12207 and Related Software Life-Cycle Standards. <*http://www.acm.org/tsc/lifecycle.html*>. Accessed 30/10/06.

MORGAN, G. 2001. *Images of Organization*. Second Edition. SAGE.

NANDHAKUMAR, J. AVISON, D.E. 1999. The fiction of methodological development. A field study of information systems development. *Information. Technology and People (ITP).* Vol.12. No.2.

NICHOLS, W.R. 2006. Building Successful Software Development Teams Using TSP and Effective Communication Networks. *CrossTalk. The Journal of Defense Software Engineering*. January.

NONAKA, I. TAKEUCHI, H. 1995. *The Knowledge-Creating Company*. Oxford University Press.

BLEICHER, P. 2003. Software Life Cycles and Kitchen Sinks. *Applied Clinical Trials Magazine*.

PELRINE, J. 2006. Cynefin - Making Sense of Agile. Agile 2006 Conference Programme. <*http://www.agile2006.org/program*>. Accessed 30/10/06.

PERKINS, D.N.T. HOLTMAN, M.P. KESSLER, P.R. MCCARTHY, C. 2000. *Leading at the Edge – Leadership Lessons from the Extraordinary Saga of Shackleton's Antarctic Expedition*. AMACOM.

PERKINS, T.K. 2006. Knowledge: The Core Problem of Project Failure. *Crosstalk. The Journal of Defense Software Engineering*. June.

PFEFFER, L. SUTTON, R.I. 2000. *The Knowing-Doing Gap*. HBS Press.

PHELAN, S.E. 1999. What is complexity science, really? *Emergence. A Journal of Complexity Issues in Organizations and Management*. The New England Complex Systems Institute.

PICHLER, R. 2006. Agile Gets Lean – How we optimized our Agile Development System using the Theory of Constraints and Scrum. *AgileDevelopment*. Spring.

PITHER, R. DUNCAN, W.R. 2006. ISO 10006 : Risky Business. <*http://www.pmpartners.com/resources/iso10006.html*>. Accessed 30/10/06.

PRIES-HEJE, J. BASKERVILLE, R.L. HANSEN, G.I. 2005. Strategy Models for Enabling Offshore Outsourcing. *Information Technology for Development*. Vol. 11. Wiley Periodicals.

RANDOM HOUSE. 2000. *Webster's College Dictionary*. Random House Publishing.

RAJLICH, V.T. BENNETT, K.H. 2000. A Staged Model for the Software Life Cycle. *IEEE Computer.*

RAYMOND, E.S. 2001. *Cathedral and the Bazaar – Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly.

REID, T.R. 2001. *The CHIP – How Two Americans Invented the Microchip and Launched a Revolution*. Random House.

RICHARDSON, J. 2005. Ancient insights into the modern organization. Chapter 20 in: *Managing Organizational Complexity: Philosophy, Theory, and Application.* A Volume in Managing the Complex:345-360.

ROONEY, D. MARTIN, N. 2003. Enabling Software Quality with Extreme Programming. *Statistics Canada Conference. Quality in IT – From Theory to Reality.*

ROYCE, W.R. 1970. Managing the development of large software systems. Proceedings IEEE WESCON. IEEE Press:1-9.

SCHILLING, M. 2005. *Strategic Management of Technological Innovation*. McCraw-Hill/Irwin.

SCHWABER, K. 2001. Agile Processes and Self-Organization. <*http://www.controlchaos.com/download/Self%20Organization.pdf*>. Accessed 30/10/06.

SCHWABER, K. 2001. *Agile Software Development with SCRUM*. Prentice-Hall.

SCHWABER, K. MARTIN, R.C. 2005. Best Practices in Scrum Project Management and XP Agile Software Development. <*www.controlchaos.com/download/ Primavera%20White%20Paper.pdf*>. Accessed 30/10/06.

SENGE, P. 1994. *The Fifth Discipline – The Art & Practice of the Learning Organization*. Currency and Doubleday.

SHAPIRO, S. 2002. The Evolution of Innovation. <*http://www.24-7innovation.com/evolution.pdf*>. Accessed 30/10/06.

SRIDHARAN, P. 2004. Visual Studio 2005 Team System. *MSDN Library*. Microsoft.

STACEY, R.D. 2001. *Complex Responsive Processes in Organizations – Learning and knowledge creation*. Routledge.

STANDISH GROUP. 2001. The Chaos Report. *The Standish Group International.*

SU, P. 2006. The World As Best As I Remember It. <*http://blogs.msdn.com/philipsu/archive/2006/06/14/631438.aspx*>. Accessed 30/10/06.

SUTHERLAND, J. VIKTOROV, A. BLOUNT, J. 2006. Adaptive Engineering of Large Software Projects with Distributed/Outsourced Teams. *Proceedings of the International Conference on Complex Systems.*

SUTTON, R.I. 2002. *Weird Ideas That Work – 11½ practices for promoting, managing and sustaining innovation*. The Free Press.

THEUNISSEN, W.H.M. 2003. *A case-study based assessment of Agile software development.* Masters thesis. University of Pretoria.

TRAURING, A. 2003. Software Methodologies – Battle of the Gurus. Info-Tech Research Group. <*http://www.fourm.info/MyArticles/SoftMeth.pdf*>. Accessed 30/10/06.

TRUCH, E. 2004. *Knowledge Orientation in Organizations*. Ashgate Publishing.

TUTU, D. 2000. *No Future Without Forgiveness*. Image.

VISE, D. 2005. *The Google Story*. Macmillan.

WEICK, K.E., SUTCLIFFE, K.M. 2001. *Managing the Unexpected – Assuring High Performance in an Age of Complexity*. Wiley/Jossey-Bass.

WEICK, K.E. 1995. *Sensemaking in Organizations*. SAGE.

WELLS, J.D. 2006. Extreme Programming. <*http://www.extremeprogramming.org*>. Accessed 30/10/06.

WHEATLEY, M.J. 2005. *Finding our way: leadership for an uncertain time*. Berrett-Koehler

WIENER, N. 1954. *The Human use of Human Beings*. Houghton Mifflin Company.

WIND, Y. CROOK, C. GUNTHER, R. 2005. *The Power of Impossible Thinking*. Wharton School Publishing.

---- o O o ----