Proceedings of the 2014 Federated Conference on Computer Science and Information Systems pp. 841–850
DOI: 10.15439/2014F196
ACSIS, Vol. 2

Situational Software Engineering
Complex Adaptive Responses of Software Development Teams

AJB (Barry) Myburgh
Jo'burg Centre for Software Engineering (JCSE)
School of Electrical and Information Engineering
University of the Witwatersrand
Johannesburg, South Africa
barrym@jcse.org.za

***Abstract*— The Complex Adaptive Situational Model (CASM) promotes understanding of establishing conditions which enable software engineering success. Influenced by complexity science, CASM explains aspects of the state of dynamic equilibrium that is achieved under constraining influence of management and production governance. Four states of dynamic equilibrium are defined: Crafted Quality (Agile), Controlled Quality (waterfall), Managed Costs (WetAgile) and Self-Directed Quality. A band of software engineering feasibility is also described and it is suggested that successful software engineering initiatives require teams to operate in that band. The journey across the band of feasibility is explained by introducing SEMAT, with Crafted Quality amounting to applying SEMAT Essence, and Controlled Quality being achieved by introducing additional practices which satisfy the more stringent governance requirements. An enterprise is then described as a collection of CAS's, thereby setting the scene for further research into the complexities of human-driven complex adaptive systems.**

I. Introduction

FOR many practitioners, Agile software development seems the best way to develop software. But old-style management often presents the biggest obstacle to successful adoption of agile approaches. The model described in this paper promotes understanding of what it takes to establish conditions which enable software engineering success, not only with agile approaches, but also traditional, plan-driven software engineering.

Humans easily relate to causal determinism – a thesis based on experience that future events (combined with the laws of nature) are sometimes the result of past and present events. For example, causal determination enables us to catch a ball by predicting in which direction it is going. Causality also enables software developers to design, plan, and predict what software will do.

Immanuel Kant (1724-1804) promoted universal causal determinism. But causality is not enough. We can't accurately predict the weather. We can't predict the full combination of features, qualities, time and resources of a software project. As explained by Jurgen Appelo: *Predictability has a devious sister called complexity* [1].

II. Complexity

Our attempts at understanding complexity involve: Dynamical systems theory; Chaos theory; Network theory; Game theory and other branches of science that are collectively known as the Complexity Sciences. Causality ruled the sciences from the 17th century. Complexity is a product of the 20th century. Complexity theory offers a new way of understanding the problem of producing software and managing organizations - even though our minds prefer causality over complexity.

The human brain is wired to find purpose and causality in everything and we favor "linear thinking" to "nonlinear thinking". So we easily reason that the global financial crisis was caused by bankers. Bad atmosphere at work is caused by the manager. The team didn't make a deadline because of someone's mistake.

The mental addiction to causal determinism has led people to use control to ensure desired outcomes. Engineers and other people with technical minds are particularly susceptible to the concept of control. Engineers developed scientific management - the command-and-control style of management. Engineers devised the kind of control systems we still find today which work adequately with repetitive tasks that don't require serious thought and analysis. But these control systems don't work with creative product development.

Managers also look for causes that would produce the outcomes exactly as they need them: through careful up-front design, with meticulous top-down planning. Appelo explains that agile management derives when hierarchical management embraces complexity and non-linear thinking and is a logical companion to Agile software development [2].

III. Challenges of Software Engineering

The software development industry started in an ad hoc way with the term "software engineering" first appearing in the 1968 NATO Software Engineering Conference where

This work was not supported by any organization

attention was given to the perceived "software crisis" of the time which resulted from the impact of rapid increases in computer power and the complexity of the problems that could be tackled. In essence, it referred (and still refers) to the difficulty of writing correct, understandable, and verifiable computer programs. The roots of the software crisis have been recognized as being complexity, expectations, and change. All too often formal approaches introduced bureaucracy and delivered software much more slowly than the rate at which requirements were changing. At the same time, some teams of passionate and disciplined programmers, with ad hoc processes and flexible requirements, delivered products of higher quality at a fraction of the cost and in a fraction of the time.

The dilemma created by the constantly high rate of software project failure in the midst of a multitude of alternative ways of working, triggered the search for general theories of software engineering that could achieve recognition equivalent to that of, for example, Maxwell's equations in the electrical engineering community [3]. But where Maxwell's equations deal with translating natural phenomena into usable practice, software engineering is all about people applying process and technology to translate their ideas into operational solutions. This translation is enabled by design which, according to John Gero and quoted by Kruchten [4], is a goal-oriented, constrained, decision-making, exploration and learning activity which operates within a context that depends on the designer's perception of the context. In the same article Kruchten explains that he had to extend the boundary of "software design" to include much more than software practitioners' traditional activities as defined in the Software Engineering Body of Knowledge (SWEBOK). In SWEBOK, software design covers only a narrow set of processes and artifacts [5]. But if we accept that design is making choices that will shape the final product, we must include some requirements activities and all coding and testing activities. The significance of this statement is that, contrary to most other engineering disciplines, the design process remains active throughout – virtually up until the very moment that source software is translated into executable machine language. And people drive the design process. As is described later in this article, people are the active agents in a complex adaptive system (CAS) and CAS agents respond to governance forces while applying rules.

An early case study that deals with the tension between approaches is described in Dee Hock's fascinating book *"Birth of the Chaordic Age"* (1999). He describes how, in the 1960's, a management team responded to their concerns when the traditional approach to system delivery was failing. *The team took ownership of the challenge and we shut ourselves in a room and didn't come out until we had an approach to which we were totally committed*. He also confirms that: *out of initial failure grew a magnificent success* [6].

In 2001 a gathering held in Utah resulted in formulation of the "Agile Manifesto" [7] which was, on the one hand, a reaction against the bureaucracy of the formal approaches, while on the other hand, also taking a stand against the "chaotic" processes and low quality products of undisciplined programmers. It gave substance to the search for a middle road between structure and non-structure, between order and chaos.

Evidence demonstrates that Agile software development, when done well, shows a tremendous return on investment. But if Agile methods have such positive effects, why doesn't everyone use them? And why are so many software projects across the world still failing? Appelo refers to a "State of Agile Development Survey 2009" [8] which identified the following factors as contributing to failed Agile approaches:

- Management opposed to change
- Loss of management control
- Lack of engineering discipline
- Team opposed to change
- Quality of engineering talent
- Organizational need for planning, predictability and documentation

This seems to suggest that management preferences are the biggest obstacles to Agile software development. The CASM model described in this article sheds further light on this contention.

In 2009 a group of leading international software engineering personalities started collaborating on an initiative to "re-found" software engineering. Ivar Jacobson (Use Cases, UML, RUP), Bertrand Meyer (Design-by-Contract and the OO Language Eiffel) and Richard Soley (CEO of the Object Management Group (OMG)) established the SEMAT Initiative - Software Engineering Method and Theory. Supporters of the initiative signed a declaration somewhat reminiscent of the Agile Manifesto and since then a great deal of work has been carried out aimed at defining the "kernel of widely-agreed elements" and often referred to as "Essence", meaning the essence of software engineering.

Figure 1 highlights two important aspects of the SEMAT kernel:

- The Areas of Concern (Customer, Solution and Endeavor) and
- The Alphas (Opportunity, Stakeholder, Requirements, Software System, Team, Work and Way of Working)

Areas of concern are addressed in terms of Activity Spaces which involve the actions taken to achieve objectives.

Alphas represent essential aspects of software engineering and each progresses through a number of states (Alpha States) as the team conducts work.

As described in the submission to the OMG [9] and in the published book *"The Essence of Software Engineering: Applying the SEMAT Kernel"* (2013) [10], the SEMAT initiative promises to fundamentally affect the discipline of software engineering.

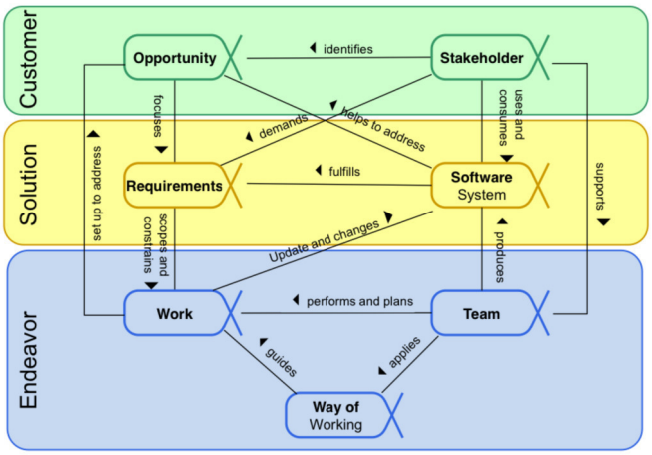

Figure 1: SEMAT Alphas in the Areas of Concern – Components of the Essence of Software Engineering

IV. Complex Adaptive Systems

A view shared by many software development experts and Agile/Lean evangelists is that software projects are complex adaptive systems (CAS's). CAS's are composed of agents - as described by M. Mitchell Waldrop in *"Complexity: the Emerging Science at the Edge of Order and Chaos"* (1992) [11]. CAS agents can be molecules, neurons, web servers, fish, starlings, and people - always forming new emergent structures with new emergent behaviors

Software projects involve people who are constantly organizing and reorganizing into larger structures: Project teams; Social groups; Task forces; Committees; etc.

CAS's are able to adapt to their environments: an infant learning to walk; car drivers evading a traffic jam; a software team adapting to what their customer really wants. Moving to the sweet spot between chaos and order, they learn and adapt and navigate their way with "chaordic" [12] processes that are neither fully ordered nor truly chaotic.

In Agile software development, we often hear reference to scientific terms such as self-organization and emergence. The concepts of emergence and the factors leading to emergent results lie at the heart of CAS theory's relevance to software development.

Examples of self-organizing systems include an ant colony, the brain, a Scrum team, a CMMI-Dev team, a team applying SEMAT Essence. Scrum, CMMI-Dev and Essence are not methodologies with defined processes or sets of procedures - they are development frameworks. And the frameworks provide rules and constraints on behavior that can cause a CAS to self-organize into an intelligent state of dynamic equilibrium.

When applying complex systems theory to software development and management, we are treating the organization as a system.

System dynamics - not to be confused with dynamical systems theory - is a technique from the 1950's to help managers understand and improve their industrial processes. System dynamics recognized that structure is often a more important contributor to an organization's behavior than individual parts themselves.

Systems Thinking was developed in the 1980's and popularized by Peter Senge's book *"The Fifth Discipline"* [13]. It's about understanding how things influence each other in the whole, a problem-solving mindset that views problems as parts of an overall system. In some ways similar to System Dynamics, but more subjective.

Social complexity is the study of complexity in social systems and to manage social complexity, we need to understand how things grow - not how they are built. This is an extension of ideas promoted by Fred Brooks in 1987 when he explained that the very essence of software engineering lies in complexity, conformity, changeability and invisibility [14].

Appelo's *"Management 3.0"* applies complexity thinking and assumes that managers cannot construct and steer a self-organizing team. The team must be grown and nurtured. Productive organizations are not managed with models and plans; they must emerge through the power of self-organization and evolution. Appelo suggests that complexity thinking is like the light that feeds all that grows and goes on to explain that at the project level, new emergent structures form and new emergent behaviors are displayed [15]. Like any other CAS with interconnected agents (people) interacting with each other to form an integrated whole. Even though software projects have many elements, only people are the real agents - the active elements. Teams themselves are agents on the next higher level.

Items that are not agents include: Requirements; Features; Artifacts; Deliverables; Tools; Technologies; Processes; Practices. They cannot actively organize and reorganize themselves. They cannot initiate interaction with any of the other elements in the project.

Appelo emphasizes that the primary focus of any manager should be to energize people - to make sure that they actually want to do what's required of them. Like a gardener looking after plants in a garden, a manager looks after the employees on the team/s [16].

For centuries mathematicians have preferred to work with linear (ordered) systems and considered nonlinear (complex) systems to be a special group. But nonlinear systems are the norm and abundant throughout the universe, whereas linear systems are a rare and special breed. From the beginning of the universe, everything in it was shaped by self-organization. Self-organization is the process where a structure or pattern appears in a system without any central authority or external element imposing it through planning. Self-organization is the norm. It is the default behavior of dynamic systems, whether these systems consist of atoms, molecules, species, businesses or software developers. Appelo emphasizes that self-organization is not a "best practice" - it is "default practice" [17]. No matter how a team

is managed, there will be self-organization. People will discuss and agree on lunch meetings, folder structures, workplace territories, birthday parties. Everything that management does not constrain - and much that it attempts to - will self-organize. Humans have behaved that way for 200 000 years.

But is what happens also happening in the "right direction"? Though every self-organizing system can have its own direction, the possible directions are limited by its environment. No self-organizing system exists without context. And the context constrains, governs and directs the organization of the system.

Environmental constraints affect the direction taken by a self-organizing system. This is illustrated by considering the Game of Life - a simple zero-player game invented in 1970 by the British mathematician John Conway. It is "played" on a grid of cells, where each cell has eight neighbors, one in each direction, including the diagonals. The cells can be born and stay alive or die as determined by the application of rules. The Game of Life is an example of a cellular automaton - a mathematical system in which cells are influenced by other cells according to some set of predefined rules. It is particularly interesting because it is a fine example of a system with a small set of simple rules, having complex behavior and ordering itself. The game also shows us that, whatever the initial situation is, the system will eventually always stabilize.

There is, however, one catch: the set of rules has to be chosen carefully. We therefore observe that rules must be tuned for a system to be both stabilizing and lively. A different set of rules leads to a different system with different behavior

As described by Waldrop [18] Stephen Wolfram proposed a classification scheme for cellular automata - named universality classes.

- Class I: These are the systems with “doomsday rules”. No matter what pattern of living and dead cells at the start, everything dies within a few generations.
- Class II: These systems are a bit livelier, but not much. Each initial pattern quickly collapses to a set of very boring, static configurations.
- Class III: These systems are at the opposite extreme: they are too lively. Each initial pattern in the system results in total chaos with no configuration stabilizing and nothing being predictable.
- Class IV: These are the systems with a set of rules not leading to dead, static, or chaotic configurations. Emerging patterns in this category are lively, creative, often surprising, but also stabilizing.

In dynamical systems, Classes I and II correspond to order. Class III corresponds to chaos. Class IV (of which the Game of Life is a famous example) corresponds to complexity. Given that complexity is usually explained as the region between order and chaos, this means that class IV finds itself between II and III.

Complex adaptive systems are systems that can find their own way toward that sweet spot of complexity, between order and chaos, where life blooms and creativity thrives. Scientists call it the edge of chaos, but they also could have called it the edge of order. This sweet spot represents a state of dynamic equilibrium between governance forces, parameters and rules that influence emergent behavior of the CAS.

Self-organization takes care of the edge of chaos when certain parameters fall within a critical range. The manager is not a game designer and is not concerned with the low-level rules of the game. Rather, the manager configures the high-level parameters, like diversity of team members, information flow between people, and connectivity between teams. When setting up governance in an organization, one responsibility of a manager is the development of a self-organizing system, defining the boundaries of the board but not the rules of the game. When a manager takes rule-making into own hands, self-organization will be significantly influenced and frustrated. And then creativity, innovation, and adaptability in the system will suffer.

Self-organization is fundamental for every complex system. But in a human social system, self-organization alone is not enough. Appelo explains how Glen Alleman described the need for management by pointing out that there is a difference between self-organizing and self-directing and this is the role of management [19]. This is not "directing" in the command and control sense. It is directing in the "required business value" sense. If self-organizing teams serve their customers, who "manages" the customer, when the customer is not prepared to behave in a "well-mannered" way? If there is more than one self-organizing team working on the same project, who coordinates the activities between these teams? When there are conflicts in resources, funding and requirements, who coordinates resolution of these conflicts? At least a little management is needed to steer self-organization in a direction that is of value to everyone in the system. Appelo points out that Sanjiv Augustine calls it "light-touch leadership". Appelo calls it alignment of constraints [20]. This author calls it balancing the governance forces.

Directed self-organization in software engineering is a matter of manipulating governance so that a group of people produces results valuable to the goals of the project.

V. The Complex Adaptive Situational Model

Humans, with the introduction of consciousness, invented morality, laws and authority. We defined preferred directions for self-organizing systems because some results are seen as valuable and others as harmful. We value human lives therefore consider malaria parasites and HIV viruses an undesirable result of self-organization. Appelo points out

that we value many irrational and unnatural things too, like non-discrimination, peace, monogamy [21]. Self-organization makes no distinction between good and bad, between virtues or vices, between valuable and harmful. Systems simply do whatever the environment allows them to do. Whatever they can get away with. And so, humans embraced the concept of command-and-control which enables attempts to steer self-organizing systems (businesses, teams, countries) in the direction that stakeholders considered to be valuable. That's how managers got their positions and how governments try to run countries. They care about results. They want to make sure that self-organizing systems either produce valuable things (products and services), or refrain from harming valuable things (human lives, economic growth, natural resources). Managers want software teams to create valuable software with which to make money or deliver good service.

Key constraints affecting the emergent behavior of a team of software engineers as a CAS are broadly identified by this author as:

- Management Governance and
- Production Governance.

Management Governance is a method or system of management practices that range from formal, high ceremony practices on the one hand, to informal, low ceremony practices on the other. The formal approach to management provides work products that could lead to high levels of visibility – producing project plan/s and progress reports, risk management plan/s and reports, quality management plan/s and reports, configuration management plan/s and status accounting reports, meeting agendas and minutes, etc. On the other hand, the informal, low ceremony approach depends less on detailed, written communication, hence leaving less visible evidence trails.

Production Governance is a method or system of production practices that range from engineering, "Waterfall" practices on the one hand, to organic, iterative or "Agile" practices on the other. A key engineering practice is to work according to the sequential stages of the life cycle, e.g.: Requirements Analysis; Design; Implementation and Unit Test; Integration and System Test; Qualification Test. Visible artifacts are then produced, including software requirement specifications, software architecture documents, software design documents, programming standard/s and test records. At the organic extreme we experience a situation where there is little emphasis on the life cycle stages and associated documentation, and high focus on the technical practices of software development.

As illustrated in Fig. 2, this author's hypothesis is that different combinations of governance constraints influence emergent behavior, resulting in four possible states of dynamic equilibrium:

- Crafted Quality (Agile)
- Controlled Quality (Waterfall or Plan-Driven)
- Managed Costs (WetAgile)
- Self-Directed Quality

While CASM identifies these four domains, it is important to realize that for a particular team and at a particular time, only one of the domains will be dynamically active to represent the emergent behavior of that team under particular circumstances.

Today's "Complex Adaptive Situational Model" (CASM) illustrated in Fig. 2 was first described as the model of "Situational Software Engineering" by Myburgh in 1992 [22]. Continuous application and research gave rise in 2005 to the second generation of the model, viz. the "Situational Process Model" (SPM), illustrating interaction between production processes and management & control processes [23]. CASM represents the third, published generation of the model and it identifies four behavioral domains that represent states of dynamic equilibrium as responses to the environmental governance constraints.

The terms "Plan-Driven" and "Agile" have been used by Boehm and Turner [24] to describe what are essentially the Controlled Quality and Crafted Quality approaches.

CASM in no way implies that the essence of software engineering is any different in Controlled Quality and Crafted Quality domains, but life cycle models will be different as later explained in this article.

A. Crafted Quality (The Curved Arrow)

This domain suits the information age organization where management formality is relaxed (low ceremony management governance) and production processes are accelerated by doing things in parallel (organic production governance). Crafted Quality is tantamount to taking an Agile approach. A key benefit of the Crafted Quality approach is faster delivery – exactly what is required in the

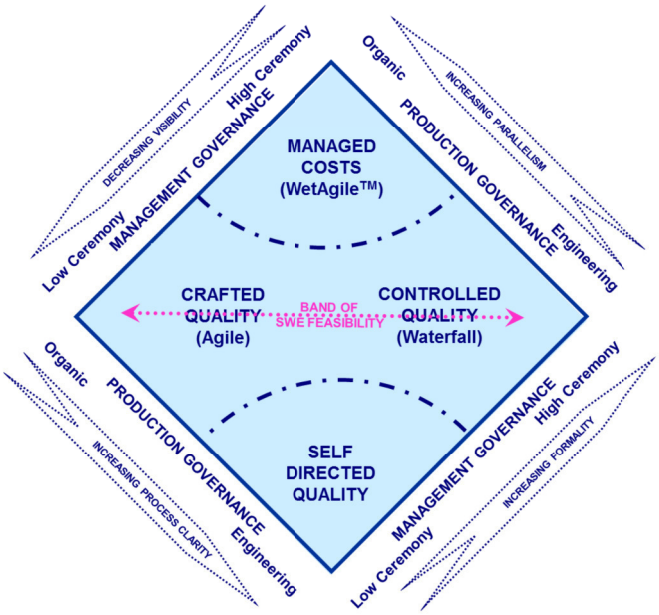

Figure 2: CASM – Insyte's Complex Adaptive Situational Model

competitive environment of the information age organization. Crafted Quality is the consequence of Agile Management meeting Agile Development.

The metaphor chosen for this domain is a curved arrow. It emphasizes the adaptive nature of an agile team that can rapidly respond to change.

But Crafted Quality does not only have benefits. Product rapidly brought to market is often not nearly defect-free, leading to potentially expensive re-work. This outcome is well demonstrated by software product with early releases that are plagued by defects that are only eradicated after a number of upgrades to the product have been implemented.

B. *Controlled Quality (The Cube)*

In this domain the emergent behavior derives from constraints of engineering-style production governance and formal, high ceremony management governance. A well-executed Waterfall approach to software engineering exemplifies Controlled Quality. Such an approach is described in the article *"They Write the Right Stuff"* by Fishman [25]. One of the key benefits of the Controlled Quality approach is that quality requirements are formally addressed at each stage of the life cycle – both in terms of initially specifying the requirements and subsequently verifying fulfillment thereof.

The metaphor chosen for this domain is the cube. It emphasizes the disciplined nature of a team operating under conditions of thorough governance.

But Controlled Quality does not only have benefits. A number of situational characteristics must apply for Controlled Quality to deliver value. These include the ability to drive out and specify requirements and having the time and other resources to analyze, specify and design the full-scope solution. And attempts to drive out full scope requirements, architecture and design can easily lead to a state of "analysis paralysis" which CASM calls "debilitating bureaucracy".

C. *The Band of Software Engineering Feasibility*

It is not by accident that CASM is represented as a diamond-shaped model. This layout places the emphasis on what is called "the band of software engineering feasibility" which stretches from Crafted Quality, Agile at the one end, to Controlled Quality, Waterfall at the other. Depending on circumstances, this implies that the state of dynamic equilibrium of a software engineering team can exist anywhere along the band and still produce value-adding results. An implied characteristic of the software engineering band of feasibility is that it is supported by a culture of "management-by-measurement", meaning that, no matter whether the way of working is Agile or Plan-Driven, management will be enabled by taking, analyzing and responding to relevant measurements.

The evolving, risk-driven approach described by Boehm in May 1988 in *"A Spiral Model of Software Development and Enhancement"* [26] could very well be understood to be a journey across the software engineering band of feasibility, with the first iteration being fully Agile, and the last solidly in Controlled Quality territory.

SEMAT Essence enables practices to define life-cycles, whether Agile or Plan-Driven, by sequencing a number of patterns, one for each phase and/or milestone in the life-cycle The life-cycles are illustrated using the template shown in Fig. 3.

Each Kernel Alpha and its states are shown in a vertical column with their creation at the top and their destruction at the bottom. Milestones are shown as a vertical bar across the grid starting with an inverted triangle to represent the milestone and continuing with a white line over which are shown the states to be achieved to successfully pass the milestone. Where achieving a state is either recommended or optional, the state is shown with a dashed outline and italicized text.

Using the template illustrated in Fig. 3, a sub-clause in the submission to the OMG provides illustrations of a few typical software engineering life-cycles, including an Exploratory Process life-cycle (Crafted Quality) and a Waterfall life-cycle (Controlled Quality). Readers who would like to review these models are urged to access the OMG submission [27].Another useful example is to be found in *"Agile and SEMAT – Perfect Partners"* [28].

This suggests that various instantiations of the SEMAT Life Cycle Model could be placed at various points across the software engineering band of feasibility.

Thus far we have considered Crafted Quality (Agile) and Controlled Quality (Plan-Driven). But what of the other two domains that are not in the band of feasibility?

D. *Managed Costs (The Explosion)*

This condition emerges when high ceremony management governance is applied to a situation that has been given the freedom of organic production governance. This means management expects Controlled Quality behavior while simultaneously giving developers the organic freedom of Agile production. A somewhat dysfunctional expectation as described in *"Corporate Information Systems Management"*

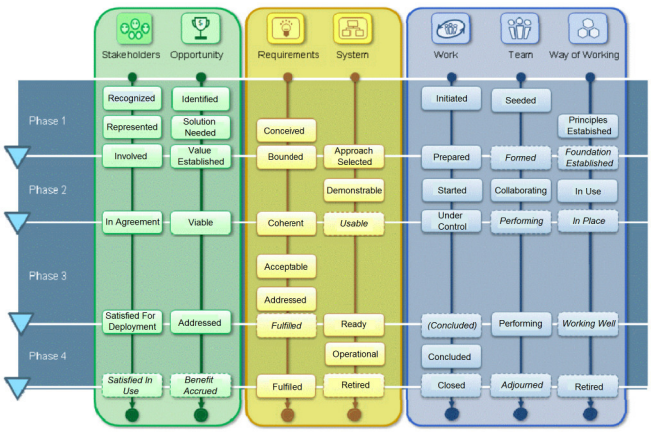

Figure 3: SEMAT Life-cycle Template

(1999) by Applegate, McFarlan and McKenney [29]. The Managed Costs name emphasizes that management will focus on cost and budget control while being quite disconnected from the day-to-day, technical activities of the team. Could this explain why Appelo's survey (referenced above) identified the following factors as contributing to failed agile approaches?

- Management opposed to change – hanging on to high ceremony management governance.
- Loss of management control – that is perceived to happen when moving to a low ceremony approach.
- Lack of engineering discipline – due to organic production approach.
- Organizational need for planning, predictability and documentation – associated with high ceremony management governance.

Steve Pieczko [30] suggests that Managed Costs might be a hybrid condition experienced by a team that is migrating from Controlled to Crafted Quality and, while still "dripping from the waterfall", they're trying to be agile. Hence the name "WetAgile", introduced in 2010 by Pieczko.

A possible explanation, but this author has experienced a number of situations where the somewhat dysfunctional, Managed Costs state seems to be permanent and a breeding ground for "management-by-politics".

The metaphor chosen for this domain is the explosion which emphasizes the often crisis-driven reality of the domain.

E. Self-Directed Quality (The Sphere)

When low ceremony management governance interacts with engineering production governance, the resulting state is Self-Directed Quality (SDQ) (The Sphere). A somewhat surprising situation. Why would practitioners elect to be constrained by engineering production governance when management governance expects no more than low ceremony? Followers of Controlled Quality would see this as an unexpected bonus, while Crafted Quality "agilista" might think of it as madness. This author suggests two possible explanations that need to be tested by means of further research.

The first might be because the tools being used enforce typical engineering production governance. It was for this reason that first generation CASM actually called this domain "Automatic Quality" [31].

A second explanation is that small, (one-person?) software development initiatives might be executed by individual/s who prefer to follow the defined stages of the engineering life-cycle. It might well be that some developers of open source software prefer to adopt this way of working.

The metaphor chosen for this domain is the sphere, emphasizing (from an engineer's point of view), the utopian situation where effective engineering is performed with few management constraints.

F. When Things Go Wrong

We earlier introduced Stephen Wolfram's proposed classification scheme where Class IV corresponds to complexity. Classes I and II correspond to order and Class III to chaos. As illustrated in Fig. 4, CASM's four domains are suggested to correspond to four types of Class IV - Complexity, with Classes I and II lying to the right of the band, and Class III to the left.

When the freedom of Crafted Quality is abused, the situation typically degenerates into a state of chaos (Class III).

Inappropriate responses to governance that desires a Controlled Quality outcome can easily result in creation of Class I or II situations with excessive order –experienced as debilitating bureaucracy.

G. CASM Characteristics

CASM has been introduced as a model of styles of team behavior and in broad terms, any software engineering team could, in response to the governance constraints imposed, be in any one of the four states of dynamic equilibrium. Each state has a set of defining characteristics. Table 1 describes characteristics of the domains. The table derives from published work (Myburgh [32], Boehm and Turner [33]) as well as from experience gained through practical application of the model.

VI. Bringing CASM To Life – Options for Management And Teams

CASM allows management to understand levels of governance that should be applied according to characteristics of the situation. The software engineering team then responds chaordically and achieves a state of dynamic equilibrium that is situationally appropriate. Table 2 identifies a number of these situational characteristics and suggests appropriate governance that should be applied. The table is based on the assumption that we are trying to pin-point the required point of dynamic equilibrium in the band

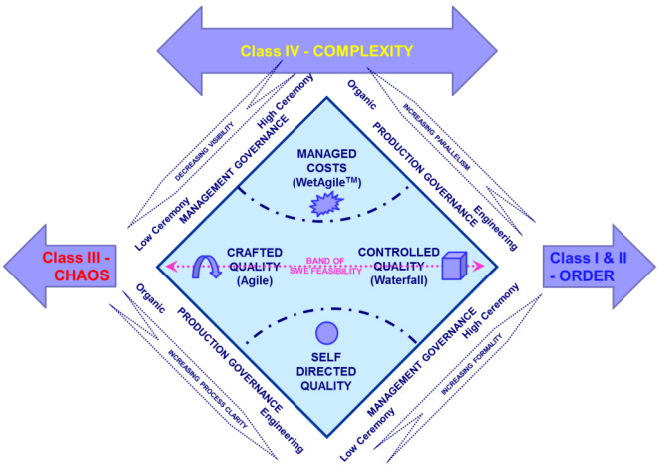

Figure 4: CASM – Modeling States of Complexity

Table 1 – CASM Domain Characteristics

	CRAFTED QUALITY (CrQ) (AGILE)	CONTROLLED QUALITY (CoQ) (WATERFALL)	MANAGED COSTS (MaC) (WETAGILE)	SELF-DIRECTED QUALITY (SDQ)
MANAGEMENT				
CUSTOMER RELATIONS	Dedicated on-site customers	As-needed, formal customer interaction	As-needed, formal customer interaction	As-needed, formal customer interaction
	Focused on prioritised increments	Focus on formal contract provisions	Focus on formal contract provisions	Focused on prioritised increments
	Best Business Practice tends to dominate (Risk-taking)	Best Solution Delivery Practice tends to dominate (Risk-avoiding)	Best Solution Delivery Practice tends to dominate (Risk-avoiding)	Best Solution Delivery Practice tends to dominate (Risk-avoiding)
PLANNING & CONTROL	Internalised plans (low visibility)	Formal, documented architecture and plans	Formal, documented architecture and plans	Internalised plans (low visibility)
	Evolutionary delivery	Incremental or full-scope delivery	Incremental or full-scope delivery	Incremental or full-scope delivery
	Qualitative control	Quantitative control	Quantitative control	Qualitative control
	Classic PMBOK practices less feasible (parallel approach)	Classic PMBOK practices more feasible (sequential approach)	Classic PMBOK practices more feasible (sequential approach)	Classic PMBOK practices less feasible (parallel approach)
	Sometimes "populate first", then plan. Meaning that the team is formed before a plan is available for the work to be done	Often "plan first", then populate. Meaning that the team is formed in response to the needs of a well-defined plan.	Often "plan first", then populate. Meaning that the team is formed in response to the needs of a well-defined plan.	Sometimes "populate first", then plan. Meaning that the team is formed before a plan is available for the work to be done
	Risk contained by time-box	Risk contained with Management Reserve	Risk contained with Management Reserve	Risk absorbed by individual
COMMUNI-CATION	Tacit interpersonal knowledge (low visibility)	Formal, documented architecture & knowledge	Formal, documented architecture & knowledge	Formal, documented architecture & knowledge
PROCESS	Innovative, "black-box", empirical processes	Deterministic, "white-box", defined process sequence	Deterministic, "white-box", defined process sequence	Deterministic, "white-box", defined process sequence
	More often unique initiatives	More often repeatable processes and continuous improvement	More often repeatable processes and continuous improvement	More often repeatable processes and continuous improvement
	Repeated initiatives remain challenging	Repeated projects can become jobs	Repeated projects can become jobs	Repeated projects can become jobs
	Could become chaotic	Usually well organised	Usually well organised	Usually well organised
TECHNICAL				
REQUIRE-MENTS	Prioritised informal stories and test cases	Formalised project capability, interface, and quality. architecture	Formalised project capability, interface, and quality. architecture	Prioritised informal stories and test cases
	Undergoing unforeseeable change	Foreseeable evolution requirements	Foreseeable evolution requirements	Undergoing unforeseeable change
DEVELOP-MENT	Evolving architecture	Guided by full-scope architecture	Guided by full-scope architecture	Evolving or full-scope architecture
	Simple design	Extensive design	Extensive design	Simple or extensive design
	Short increments	Longer increments	Longer increments	Short increments
	Re-work assumed inexpensive	Re-work assumed expensive	Re-work assumed expensive	Re-work assumed expensive
TESTING	Executable test cases define requirements	Documented test plans and procedures	Documented test plans and procedures	Formal test plans and procedures
PERSONNEL				
CUSTO-MERS	Dedicated, collocated CRACK performers	CRACK performers, not always co-located	CRACK performers, not always co-located	Dedicated, collocated CRACK performers
DEVELOP-ERS	Led by those who revise methods to meet situation	Less involvement of those who revise methods	Less involvement of those who revise methods	Less involvement of those who revise methods
	Learn largely by doing	Learn largely by reading	Learn largely by reading	Learn largely by reading
CULTURE	Many degrees of freedom	Framework of policies and procedures	Framework of policies and procedures	Self-limited degrees of freedom
	Thriving on chaos	Thriving on order	Thriving on order	Thriving on order
APPLICATION				
PRIMARY GOALS	Rapid value	Predictability, Stability	Predictability, Stability	Predictability, Stability
	Responding to change	High assurance	High assurance	High assurance
SIZE	Smaller teams and projects	Larger teams and projects	Larger teams and projects	Smaller teams and projects
ENVIRON-MENT	Turbulent	Stable	Stable	Turbulent or stable
	High change	Low change	Low change	High or low change
	Project-focused	Project/organisation focused	Project/organisation focused	Project-focused

CRACK = Collaborative, Representative, Authorised, Committed, Knowledgeable

Table 2 – Appropriate Software Engineering Responses to Situational Characteristics

SITUATIONAL CHARACTERISTIC	YES THIS DESCRIBES THE SITUATION	NO THIS DOES NOT APPLY
Are requirements readily definable?	A CoQ, Waterfall way of working could be adopted on condition that delivery time-scales permit.	The CrQ, Agile way of working is required.
Is there a comprehensive architectural description for the solution?	A CoQ, Waterfall way of working could be adopted. If the scope of delivery is large, an incremental approach will mitigate risk by delivering regular, pre-planned increments.	The CrQ, Agile way of working is required.
Is there pressure to rapidly produce results?	The CrQ, Agile way of working is required.	A CoQ, Waterfall way of working could be adopted on condition that requirements are definable.
Is there pressure to produce accurate schedules, budgets & estimates?	If the accuracy is to be based on schedules, budgets and estimates that are derived from a detailed action plan for the initiative, then a CoQ, Waterfall approach is required. If the accuracy is to be based on the cost per time-box, then a CrQ, Agile approach is indicated.	Schedules, budgets and estimates can be based on the cost per time-box and a CrQ, Agile way of working is suggested.
Does the size of the initiative introduce significant risk?	A CoQ, Waterfall approach is suitable for mitigating this risk – on condition that other characteristics required for CoQ also pertain.	The CrQ, Agile way of working is suggested so that the overhead associated with CoQ, Waterfall can be avoided.
Will implementation of the solution introduce significant change?	A CrQ, Agile way of working allows for resistance to change to be mitigated by limiting the extent of change associated with each iteration. An incremental, CoQ, Waterfall approach could also be used to limit the extent of change introduced during each increment.	Either CoQ, Waterfall or CrQ, Agile approaches could be viable. Other situational characteristics will influence the decision.
Is there significant risk due to technology? (This suggests that unproven, state of the art technology is to be implemented).	A CrQ, Agile approach should be followed by a team that is mandated to experiment with and get to know the new technology.	Either CoQ, Waterfall or CrQ, Agile approaches could be viable. Other situational characteristics will influence the decision.
Does cost-of-failure represent a source of significant risk?	A CoQ, Waterfall way of working should be adopted to allow for product assurance. If the scope of delivery is large, an incremental approach will further mitigate risk by delivering regular, pre-planned increments that could be separately assured.	Either CoQ, Waterfall or CrQ, Agile approaches could be viable. Other situational characteristics will influence the decision.
Does the software engineering team collectively have a high level of competence?	The team should be able to adapt to whatever way of working is situationally appropriate.	This situation represents a significant source of risk, and attempts to adopt a CoQ way of working could easily result in "debilitating bureaucracy", whereas CrQ approaches are likely to evolve into "freelance chaos".

of feasibility. Hence suggestions made refer only to options that lie in this band.

VII. Bringing CASM To Life – Options For The Enterprise

The above analysis of situational factors demonstrates that different ways of working apply to different situations. A small team of software engineers who are working on a focused initiative can be expected to adjust their way of working to be appropriate to the situation. However, in larger organizations where many teams are tackling many initiatives, one could expect different teams to be in different states of dynamic equilibrium at the same time.

To better understand this, we can consider the idea of a hierarchy of people-based complex adaptive systems – a system of complex adaptive systems. Working from the bottom up, we first find an individual person. (Remembering that a single person is already a CAS). If a few people collaborate towards achieving the same goal/s, we discover the next level CAS, viz. a team. Teams could also be contributing to achievement of common goal/s and hence a collection of teams could define the next higher level. For the purpose of this discussion, the highest level CAS will be the enterprise itself. Now, by employing the various metaphors associated with each state of dynamic equilibrium, the diagram in Fig. 5 represents an enterprise as a collection of complex adaptive systems and the metaphor for the Enterprise is suggested to be an amoeba.

VIII. Conclusion

The Complex Adaptive Situational Model described in this paper promotes understanding of what it takes to

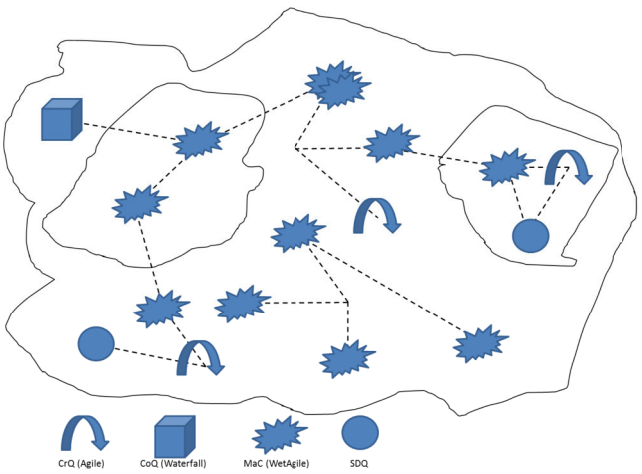

Figure 5: The Enterprise "amoeba" - represented as a System of Complex Adaptive Systems

establish conditions which enable software engineering success, not only with agile approaches, but also traditional, plan-driven software engineering.

Influenced by complexity science, CASM explains aspects of the state of dynamic equilibrium that is achieved by a software engineering team under the constraining influence of management and production governance.

Four states of dynamic equilibrium are defined: Crafted Quality (Agile), Controlled Quality (Waterfall), Managed Costs (WetAgile) and Self-Directed Quality. A band of software engineering feasibility is also described and successful software engineering initiatives require teams to operate in that band which stretches from Crafted Quality to Controlled Quality. Management's challenge is to apply appropriate governance to enable the required state of dynamic equilibrium.

The journey across the band of feasibility is further described by introducing SEMAT, with Crafted Quality amounting to applying SEMAT Essence, and Controlled Quality being achieved by introducing additional practices which satisfy the more stringent governance requirements.

CASM in its four states then allowed introduction of the idea of describing an enterprise as a collection of complex adaptive systems, thereby setting the scene for further research into the complexities of human-driven complex adaptive systems.

ACKNOWLEDGMENT

Many thanks to Dr. Alastair Walker for early support of the model and coining the name of the *Managed Costs* domain. Thanks also to Dr. Barry Dwolatzky for continued support and the opportunity to further develop the model. Dr. Whitey van der Linde is thanked for helping to find the links between the model and complexity science. The substance that SEMAT gives to Situational Software Engineering is primarily thanks to Dr. Ivar Jacobson, one of the founders of SEMAT.

Jurgen Appelo receives special thanks for the overview of complexity and complex adaptive systems that is at times paraphrased from: *"Management 3.0 – Leading Agile Developers and Developing Agile Leaders"* (2011). Readers of this article are encouraged to also read Appelo's publications [http://www.mgt30.com/].

Thanks also to other reviewers of the article including Adrian Schofield, Steve Piezcko, Paul MacMahon. Their feedback improved the final product.

REFERENCES

[1] J. Appelo, Management 3.0 – Leading Agile Developers, Developing Agile Leaders, Addison-Wesley, 2011, p. 2.
[2] J. Appelo, Management 3.0 – Leading Agile Developers, Developing Agile Leaders, Addison-Wesley, 2011, p. 11.
[3] Johnson, P, Ekstedt, M, Jacobson, I, "Where's the Theory for Software Engineering?,"http://dx.doi.org/10.1109/MS.2012.127, pp. 94-96.
[4] P. Kruchten, "Casting Software Design in the Function-Behavior-Structure Framework,"http://dx.doi.org/10.1109/MS.2005.33, pp. 52-58.
[5] A. Abran et al., eds., Guide to the Software Engineering Body of Knowledge, IEEE CS Press, 2004.
[6] D. Hock, Birth of the Chaordic Age, Berrett-Koehler Publishers, Inc., 1999, pp. 205-207.
[7] "Agile Manifesto," [Online]. Available: http://agilemanifesto.org/. [Accessed 15 January 2014].
[8] J. Appelo, Management 3.0 – Leading Agile Developers, Developing Agile Leaders, Addison-Wesley, 2011, p. 28.
[9] OMG, "Essence - Kernel And Language For Software Engineering Methods 1.0 - Beta 1," July 2013. [Online]. Available: http://www.omg.org/spec/Essence/1.0/Beta1/.
[10] I. Jacobson, P.-W. Ng, P. E. McMahon and I. Spence, The Essence of Software Engineering - Applying the SEMAT Kernel, http://dx.doi.org/10.1145/2380656.2380670.
[11] M. M. Waldrop, Complexity: the Emerging Science at the Edge of Order and Chaos, http://dx.doi.org/10.1063/1.2809917, p 145.
[12] D. Hock, Birth of the Chaordic Age, Berrett-Koehler Publishers, Inc., 1999, p. 3
[13] P. Senge, The Fifth Discipline - The Art and Practice of the Learning Organization, http://dx.doi.org/10.1108/eb025496.
[14] F. P. Brooks, Jr., "No Silver Bullet - Essence and Accidents of Software Engineering", http://dx.doi.org/10.1109/MC.1987.1663532.
[15] J. Appelo, Management 3.0 – Leading Agile Developers, Developing Agile Leaders, Addison-Wesley, 2011, pp. 50-51.
[16] J. Appelo, Management 3.0 – Leading Agile Developers, Developing Agile Leaders, Addison-Wesley, 2011, p. 58.
[17] J. Appelo, Management 3.0 – Leading Agile Developers, Developing Agile Leaders, Addison-Wesley, 2011, p. 100.
[18] M. M. Waldrop, Complexity: the Emerging Science at the Edge of Order and Chaos, http://dx.doi.org/10.1063/1.2809917, pp 225-226.
[19] J. Appelo, Management 3.0 – Leading Agile Developers, Developing Agile Leaders, Addison-Wesley, 2011, p. 100, p. 153.
[20] J. Appelo, Management 3.0 – Leading Agile Developers, Developing Agile Leaders, Addison-Wesley, 2011, p. 100, p. 154.
[21] J. Appelo, Management 3.0 – Leading Agile Developers, Developing Agile Leaders, Addison-Wesley, 2011, p. 100, p. 101.
[22] A. J. B. Myburgh, "Successful Combinations of Software Engineering Strategy and Project Management," in Proceedings of the SAIEE Symposium "Professional Issues in Software Project Management - 5 September 1990", Johannesburg, 1992.
[23] A. J. B. Myburgh, "Towards Understanding The Relationship Between Process Capability And Enterprise Flexibility," in Proceedings of "SAATCA 8th International Systems Auditor Convention 24 – 25 August 2005", Johannesburg, 2005.
[24] B. Boehm and R. Turner, Balancing Agility and Discipline – A Guide for the Perplexed, Addison-Wesley, 2004.
[25] C. Fishman, "They Write the Right Stuff" 2007. http://www.fastcompany.com/magazine/06/writestuff.html, Last Accessed 22 June 2014.
[26] B. W. Boehm, "A Spiral Model of Software Development and Enhancement", http://dx.doi.org/10.1109/2.59, pp. 61-72.
[27] OMG, "Essence - Kernel And Language For Software Engineering Methods 1.0 - Beta 1," July 2013. [Online]. Available: http://www.omg.org/spec/Essence/1.0/Beta1/, p 267-271.
[28] I. Jacobson, I. Spence and P. Ng, Agile and SEMAT – Perfect Partners, http://dx.doi.org/http://dx.doi.org/10.1145/2380656.2380670..
[29] L. M. Applegate, F. W. McFarlan and J. L. McKenney, Corporate Information Systems Management - 5th Edition, McGraw-Hill, 1999, p. 184.
[30] S. Pieczko, "Waterfall? Agile? How About WetAgile?", 2010. http://www.WetAgile.com, Last Accessed 18 June 2014.
[31] A. J. B. Myburgh, "Successful Combinations of Software Engineering Strategy and Project Management," in Proceedings of the SAIEE Symposium "Professional Issues in Software Project Management - 5 September 1990", Johannesburg, 1992, p. 94.
[32] A. J. B. Myburgh, "Towards Understanding The Relationship Between Process Capability And Enterprise Flexibility," in Proceedings of "SAATCA 8th International Systems Auditor Convention 24 – 25 August 2005", Johannesburg, 2005.
[33] B. Boehm and R. Turner, Balancing Agility and Discipline – A Guide for the Perplexed, Addison-Wesley, 2004.