

https://dspace.org/

Utilising the Software Engineering Methods and Theory framework to critically evaluate software engineering practice in the South African banking industry

Le Roux, Alistair Graham

2016-03-17
http://hdl.handle.net/10539/20117

Downloaded from a DSpace repository. https://dspace.org/

Utilising the Software Engineering Methods and Theory framework to critically evaluate software engineering practice in the South African banking industry

---

Alistair Graham Le Roux

agleroux@gmail.com

A research report submitted to the Faculty of Engineering and The Built Environment of the University of Witwatersrand, Johannesburg

In partial fulfilment of the requirements for the Degree of Master of Science in Engineering

Supervised by Professor Barry Dwolatzky

September 2015

Declaration

I declare that this research report is my own, unaided work, except where otherwise acknowledged. It is being submitted for the Degree of Master of Science in Engineering in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree in any other university

_____

Alistair Graham Le Roux

Signed this ____ day of _____ 20____

Acknowledgements

Professor Barry Dwolatzky for his supervision, guidance and encouragement throughout the course of this research project.

Mr Johannes Vorster for his insight and encouragement in the initial phases of this project.

Abstract

In recent years, software has become the cornerstone of banking and new business products are directly dependant on software. The delivery cycles for new features is now related to market share. This drive to use software as a vehicle for competitive advantage has created an environment in which software development of new business systems are increasingly on the critical path of many projects. An organisation's portfolio of software intensive projects is situated within this complexity and organisations attempt to mitigate the risks associated with these complexities by implementing software development processes and practices. A key problem facing the modern bank is how to define and build a software development process that caters for both the traditional and increasingly agile genres of software development characteristics in a consistent and manageable way.

The banks attempt to address this problem through continuous methodology and process improvements. Comparing and assessing non-standardised software engineering lifecycle models without a common framework is a complex and subjective task. A standardised language is important for simplifying the task for developing new methods and practices or for analysing and documenting existing practices.

The Software Engineering Methods and Theory (SEMAT) initiative has developed a standardised kernel of essential concepts, together with a language that describes the essence of software engineering. This kernel, called the Essence, has recently become an Object Management Group (OMG) standard. The Essence kernel, together with its language, can be used as the underpinning theory to analyse an existing method and help provide insights that can drive method enhancements.

The research report proposes a simple, actionable analysis framework to assist organisations to assess, review and develop their software engineering methods. The core concepts of the methodology are identified and mapped to the Essence concepts. The governance model of the Essence is mapped to the governance model of the industry model and a set of practices is identified and documented in the Essence language. The mapping and resulting analysis can be used to test the validity of the Essence theory in practice and identify areas for improvement in both the method and the Essence standard.

The analysis framework has been applied to an operational software development lifecycle of a large South African bank. A mapping of the Essence concepts to the governance model and method documented in the lifecycle was completed. This mapping revealed that the Essence is a valid tool and can be used to describe a method in practice. Furthermore it is useful as an analysis framework to assess the governance model that manages and measures the progress of an endeavour in the Bank.

The case study and resulting analysis demonstrate that the Essence standard can be used to analyse a methodology and identify areas for improvement. The analysis also identified areas for improvement in the Essence specification.

Table of Contents

Declaration ii
Acknowledgements iii
Abstract i
Table of Contents ii
List of Figures iii
List of Tables iii
Glossary iv
Chapter 1 Introduction 1
Chapter 2 Literature Review 3
Defining Software Engineering 3
Factors and Challenges that Software Engineering must solve 4
Software Engineering Process Models 5
Software Engineering Theories and Standards 6
Defining Software Development Practices 8
Method Languages and Model Comparisons 8
The Value of SEMAT 8
Chapter 3 Research Objectives 10
The Research Question 10
The Research Approach 10
The Analysis Framework 11
The Case Selection 11
The Data Collection 11
Chapter 4 Research Theory and Data 13
SEMAT Essence Overview 13
The Bank's Model 18
Identifying the Bank's Alphas 23
Identifying the Bank Alpha States 24
Mapping the Bank's Alphas to the Essence Alphas 26
Mapping the Essence States onto the Bank's Lifecycle 28
Mapping the Essence Activity Spaces to the Bank's Lifecycle 29
Identification of Practices using the Essence definition of a Practice 30
Chapter 5 Analysis of the Research 32
Analysis of the Essential Concepts 32
Analysis of Practices 35
Analysis of Activity Spaces 35
Chapter 6 Conclusion and Future Work 37
Appendix A: Detailed Analysis of Observations 40
A1: Analysis of Essential Concepts 40
A2: Analysis of Essence Alpha States 43
A3: Analysis of Activity Spaces 44
A4: Analysis of Baselines 45
Appendix B: Bank's Alpha Table 46

Appendix C: Mapping Slots to Baselines 49
Appendix D: Mapping CAR's to Slots 50
References 52

List of Figures

Figure 1: The fundamental goal of software engineering 3
Figure 2: Core components of the Essence 14
Figure 3: Essence kernel alphas 14
Figure 4: Alpha decomposition 15
Figure 5: Essence activity spaces 16
Figure 6: Essence components of a practice 17
Figure 7: Phase pattern 18
Figure 8: The Bank's SDLC elements 18
Figure 9: Key SDLC language concepts 19
Figure 10: Method metamodel 20
Figure 11: Phases of the management model 21
Figure 12: Governance model 22
Figure 13: Conceptual model of a slot 23
Figure 14: Information contained in a work product 25
Figure 15: Slots modelled as kernel elements 34
Figure 16: Activity spaces mapped to activities 35

List of Tables

Table 1: Slots linked to gates 24
Table 2: Bank alphas identified in work products 24
Table 3: Slot states for a gate 25
Table 4: States for non-slot alphas 26
Table 5: Slots mapped to Essence alphas 27
Table 6: Alphas mapped to non-slot alphas 28
Table 7: Essence alpha states mapped to baselines 28
Table 8: Essence alpha states mapped to full lifecycle 29
Table 9: Activity spaces per phase per alpha 30
Table 10: Component modelling practice 31
Table 11: Analysis of slots and alphas 42
Table 12: Analysis of Essence alpha states 43
Table 13: Analysis of activity spaces 44
Table 14: Analysis of baselines 45
Table 15: Banks alpha table 46
Table 16: Mapping slots to baselines 49
Table 17: Mapping CAR's to slots 51

Glossary

Activity	An activity describes some work to be performed
Activity Space	A set of essential things to do in a software endeavour
Agile Manifesto	A formal proclamation designed to guide an iterative and people-centric approach to software development
Alpha	An alpha is an essential element of a software engineering endeavour
CSR	Conceptual solution review
Baseline	A milestone set by the governance gates in the Bank's lifecycle
BRR	The business requirements review of the Bank's lifecycle
BTF	The build, test and fix phase of the Bank's lifecycle
Business Context	A document describing a business opportunity
CAB	Change advisory board which manages all changes to production systems
CAR	A continuous assessment criteria used to assess the state of a work product in a governance gate review
CDR	The critical design review, which assesses the detail designs of a software endeavour
Competency	The essential things to know in a software engineering endeavour
DD	The detailed design phase of the Bank's lifecycle
Deployment Unit	A deployable piece of software, data or infrastructure
Discipline	A list of tasks that are grouped as part of a domain
EAPR	The enterprise architecture peer review of the Bank's lifecycle
Essence	A standard that defines the essential set of concepts that are common to all software projects
Essence kernel	The essential set of alphas and activity spaces that are common to all software projects
Functional Component	A concept that represents the static structure and dynamic behaviour of an element of the software to be built
Function-Behaviour-Structure	An ontology that represents the process of designing as transformations between function, behaviour and structure
FRF	Financial review forum, where project budgets are authorised
HLD	The high level design phase of the Bank's lifecycle, where the overall solution is defined
IDA	The infrastructure design authority, which reviews infrastructure designs
IEEE	The Institute for Electronics and Electrical Engineers
IMPL	The implementation phase of the Bank's lifecycle
ISO 24744	An ISO standard for software engineering metamodeling
LSR	Logical solution review, where the high level solution architecture is reviewed
MD	The macro design phase of the Bank's lifecycle, where the software components data and interfaces are defined
Metamodel	A model that defines the underlying concepts of a model
Node	A logical container with a defined set of attributes that supports software, data and infrastructure
Object Management Group	An international standards consortium
OPEN	Object-oriented process, environment and notation. A public domain, fully object-oriented methodology and process
Open Process Framework	A public-domain object-oriented framework of method components
Operational Component	A concept that represents the non-functional qualities required to support an element of the software to be built
Pattern	A pattern is a generic mechanism for naming complex concepts that are made up of several Essence elements
PBR	Project benefits review
PDR	The preliminary design review, which assesses high level application design
PIR	Post implementation review
Practice	A practice is a description of how to handle a specific aspect of a software engineering endeavour
PRR	The production readiness review, which assesses the production release readiness of the software and infrastructure
SDLC	System Development Lifecycle which describes a process for planning, creating, testing, and deploying a software system
SEMAT	Software Engineering Methods and Theory. Founded in September 2009 by Ivar Jacobson, Bertrand Meyer, and Richard Soley
Slot	A conceptual placeholder for a set of work products to be reviewed at a governance gate

SO	The solution outline phase of the Bank's lifecycle
Sociotechnical	A concept that recognizes the interaction between people and technology
SOMA	Service oriented modelling architecture, a practice for developing a service oriented architecture
SPEM (2.0)	Systems Process Engineering Metamodel, an OMG standard for modelling software engineering processes, currently at version 2.0
SRR	The system requirements review, which reviews the system use cases and infrastructure requirements of a solution
TRB	The technical review board, which owns and manages the six technical governance gates in the Bank's lifecycle
TRR	The testing readiness review, which reviews the system and user acceptance test cases and test environment readiness
UD	The micro design phase in the Bank's lifecycle, where the detailed internal aspects of an application is designed
Work Products	An output of an activity, typically a document

Chapter 1 Introduction

As software has become the cornerstone of banking models in recent years a situation has arisen where new business products are directly dependant on customer facing software. The delivery cycles for new features in the software frontends is now related to the market share of the product. This drive to use software as a vehicle for competitive advantage has created an environment in which software development of new business systems are increasingly on the critical path of many projects. These projects are expected to deliver quality, flexible, low cost systems in significantly shorter periods of time. This environment is further complicated by a rapid pace of technological change, increased end user involvement, outsourcing of non-core business domains to vendors, an increasing reliance on off-the-shelf software and a requirement to interoperate across organisational boundaries [1].

The turnaround time for this pace of software delivery requires refactored software development processes and lifecycles that support fast time to market, rapidly changing requirements, unknown user requirements and heterogeneous deployment environments. The frontend systems, however, still require integration with the large, well understood back end information and transaction processing systems that are the heart of business [2]. These systems are large, complex, largely undocumented and require a different software process. They need a process that allows for careful requirements analysis, detailed systems analysis, and impact analysis with clearly specified changes. These changes need to be rigorously tested through processes that include regression and integration testing.

Furthermore organisations are beginning to realise that these complex systems are core assets to the organisation and are demanding that they are modified in a structured, repeatable and cost effective way[3]. An organisation's portfolio of software intensive projects is situated within this complexity and organisations attempt to mitigate the risks associated with these complexities by implementing software development processes and practices. These methodologies prescribe a set of well-defined activities and tasks designed to provide structure, repeatability and stability to the organisation's delivery capabilities. These robust processes and practises are generic and often ignore the social, organisational and problem context of the organisation [4][5][6].

Alter describes software projects as a type of work system, and identifies nine elements (customers, products and services, processes and activities, participants, information, technologies, environment, infrastructure, and strategies) that are linked in a multi-dimensional model that is required to understand and model any system[7]. These elements need to be defined in a way that enables an efficient work system that achieves the objectives of all role players.

A key problem facing these organisations is how to define and build a software development process that caters for both the traditional and increasingly agile genres of software development characteristics in a consistent and manageable way. These processes must be defined in a way that does not neglect the systemic interrelationships that have been cultivated between the operations of the organisation and the technology systems supporting them. Ignoring the work systems that frame the software development context will reduce the amount of agility the organisation can tolerate[8].

The de facto runtime lifecycles of the banks in South Africa are hybrid models that attempt to balance this desire for greater agility with the operational risk that is their reality. Analysing these models for improvement is challenging because:

- Lifecycle terms and definitions are inconsistent and have many definitions[9]
- Project management, system development and software development lifecycles are tightly coupled to the point that it is no longer clear to which domain a term or definition belongs[10]
- Linking agile and plan driven models to a formal program and portfolio management framework is difficult as the core concepts across the models are neither standardised nor consistent[11][12].

Furthermore, a single development technique is not appropriate in these large heterogeneous environments. Applying object oriented techniques to sequential programing paradigms rooted in structured analysis is inappropriate and confusing, leading to teams not adopting a method or set of techniques because it is unhelpful in the analysis of their system[13].

Software engineers and the organisations that employ these process models require an effective thinking framework, to bridge the gap between their current ways of working and any new ideas and practices that they need to adopt. Senge proposes this model as a way to ensure a systems approach to thinking about problems[14].

Software Engineering Methods and Theory (SEMAT) as a framework has proposed a model that standardises the core elements of any software development model. SEMAT is an initiative that is

"attempting to re-found software engineering based on a solid theory, proven principles and best practices that:

- *Include a kernel of widely-agreed elements, extensible for specific uses*
- *Addresses both technology and people issues*
- *Are supported by industry, academia, researchers and users*
- *Support extension in the face of changing requirements and technology"* [15]

SEMAT provides a thinking framework that can help organisations with large, complex software development models to begin streamlining and improving their models by using a standardised framework with a core set of elements. SEMAT has managed to define the core elements of a software development endeavour and the lifecycle of these elements without prescribing a process that must be followed. This provides organisations with a powerful, standardised and paradigm independent language to help align actual project processes and techniques and the formally defined processes required by management.

The remainder of the research report is structured as follows. Chapter 2 presents a literature review of related work. Thereafter, Chapter 3 details the research objectives, design and methodology approach. Chapter 4 presents the research work completed and is structured into three sections, a brief introduction to the theory of SEMAT, a description of the Bank's model and a mapping of the models using the methodology described in Chapter 3. Chapter 5 provides an analysis and discussion of the research work that was completed and Chapter 6 concludes the research report and discusses possible future work.

Chapter 2 Literature Review

In the review of available literature the following themes emerged:

- Defining software engineering
- Factors and challenges that software engineering must solve
- Software engineering process models
- Software engineering theories and standards
- Method languages and model comparisons
- Defining software development practices
- The value of SEMAT

The chapter discusses each of the themes briefly to give the reader a broader view of software engineering as it relates to models, processes, practices and theories. The chapter concludes with a section on the value that the SEMAT initiative brings to the current body of knowledge.

Defining Software Engineering

A survey of available literature to identify a definition for software engineering revealed that it is a broad domain covering a range of topics. The IEEE defines software engineering as:

"(A) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (B) The study of approaches as in definition (A)."[16]

Sommerville, like many software engineering textbook authors, limits the definition of software engineering to software development and software maintenance:

"Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification to maintaining the system after it has gone into use."[17]

Somerville's definition leads to the idea that software engineering is about achieving the overall basic goal of taking a user's need through a process and delivering working software that satisfies this need. This basic concept is illustrated here in a diagram adapted from[18]:

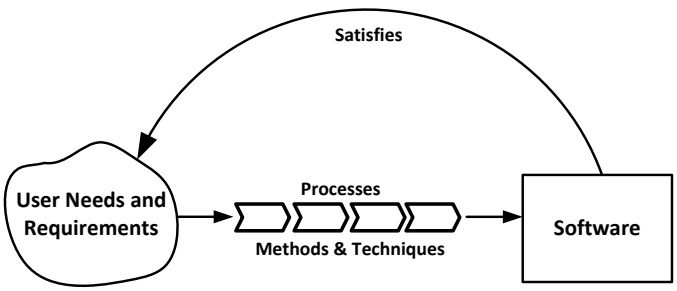


Figure 1: The fundamental goal of software engineering (adapted from [18])

All the available literature supports Somerville's definition and suggests that the essential aspects required for software engineering to achieve its basic goal are:

- Requirements gathering
- Software design and implementation
- Validation of the software against the requirements.

This view does not explicitly consider the sociotechnical nature of software engineering as an essential aspect. Furthermore it is evident that there is wide disagreement on the processes, methods and techniques required to effectively and efficiently achieve these tasks[19].

The IEEE definition of software engineering is adopted for this paper with the explicit inclusion of the sociotechnical aspects of a software engineering endeavour. This definition is consistent with the SEMAT Essence kernel which considers the sociotechnical aspects of software engineering as essential.

Factors and Challenges that Software Engineering must solve

McLeod et al [1] discusses how software development in industry today is faced with factors that are challenging the current theories about how to effectively build software intensive systems. Today's theories must be able to deal effectively and efficiently with the rapid pace of technological change. This change quickly renders software development languages and techniques obsolete and makes it increasingly difficult to manage development skills. The increasing sophistication of development tools and technologies facilitates faster development turnaround times, but introduces architecture and maintenance issues [20].

The current trend of organisations to focus on their core business functions and capabilities has led to greater reliance on packaged software acquisition rather than bespoke development[21]. These packages require customisation and integration into the organisations system landscape and have paved the way for greater outsourcing as deep domain knowledge now resides with the software package vendor and is not required in the technology department. Furthermore, there is an increased emphasis on enterprise wide systems that allow for standardised, streamlined and cost effective business processes that increases the complexity of the requirement gathering and design processes required[21]. These factors are interrelated and together demand different approaches to software engineering and provide the potential for unpredictable or unintended consequences.

Bloch et al [22] discuss four ways to improve IT project delivery and identify factors that contribute to project failures:

- Missing focus (vague vision and weak requirements)
- Content issues (incorrect details and missing content)
- Skill issues (developer and project management skills)
- Execution issues

They determined that managing strategy and stakeholders, the right technical experts, an effective team and core project management practices are the key factors for success and are the cornerstones of any project. They also propose that mastering technology by acquiring the right technical experts will somehow ensure that the right technical processes are adopted.

Furthermore, organisations are faced with significant challenges in introducing agile practices into existing traditional plan driven models and processes. Merging these models into a consistent model that does not reduce agility or ignore the existing working processes is difficult as the core concepts across the models are neither standardised, consistent nor necessarily organisationally suitable[11]. Additionally, project management, system development and software development lifecycles are tightly coupled and there is no clear distinction between the concepts that are defined for governance, project management coordination and structuring, and the concepts specifically required for developing software[23].

Software Engineering Process Models

The modern software development lifecycle has become complex as customers and stakeholders demand newer entry points into the lifecycle as their demands are not always for a new software solution. The modern lifecycle needs to balance enterprise architecture strategies, innovative business solutions, stability of operations, outsourcing, and integration and multiple competing sponsors and priorities. The modern lifecycle can be considered 'messy' [24] and will depend on coordination and synchronisation more than individual productivity.

Current trends in software engineering are revealing the importance of systems thinking and the complex nature of project teams and the critical importance of understanding an organisation's culture, project team members, their tasks and processes and how these variables interact, synchronise and develop emergent behaviour [5]. Appelo illustrates how project teams interact as agents and require significant management input to synchronise, and, the processes and practices that help teams to synchronise effectively are more likely to be adopted. Myburgh [25] extends this idea and proposes that a software engineering endeavour is a combination of management governance and production governance. He postulates that when management and production processes are appropriately synchronised then software engineering enters a band of feasibility where there is a high probability that good quality software can be delivered within budget and on time. This band spans across both agile and plan driven approaches. Kajko-Mattsson [20] supports the distinction between engineering and management processes and identifies an overriding process within which other processes branch off. These overriding processes are management processes and are the lifecycle elements of a software development lifecycle and the methods are the engineering aspects.

Boehm[8] postulates that neither agile nor rich process models solve all software engineering problems and do not provide any silver bullets. He observes that, though methods are useful, more value will be found in focussing on people and communication. He recommends that while agile and plan driven approaches have strong home grounds, strategies are needed to integrate them to capitalise on their strengths and minimise the effects of their weaknesses. Each organisation needs to balance their approach to software engineering to fit the organisations evolving needs and not assume that an engineering paradigm that worked previously will work into the future.

Many of the rich software engineering process models and standards in use today are rooted in the idea that software processes are software too. This idea was first proposed by Osterweil in the late Eighties [26]. Kruchten[27] argues that these rich processes collapse due to their highly prescriptive artefact based foundations. He questions whether these process models are addressing the right problems and that process defined as executable software is the wrong metaphor and incorrectly

assumes that human actors are merely machines blindly executing process. He expresses the view that we should not start with complex, detailed all-encompassing processes before we have identified the right paradigm. Peter Senge [14] confirms this view about the hazards of applying fixes to effects without understanding the true causes and the context of those causes in his work on systems thinking. Kruchten postulates that we need a descriptive approach based on observing what people do rather than what we think people should do.

Kajko-Mattsson [20] describes how homogenous project processes are not optimised to run across multiple teams and environments. To embrace this heterogeneity, software organizations have created many process variants that are dedicated to various needs, development styles, product complexities, process formalities, cultures, and project types. Each project team will attempt to satisfy its specific needs while fitting into the organisation's overarching prescriptive processes by creating a project process. This results in multiple variants of processes and practice across teams and introduces productivity constraints and significant synchronisation issues. Kajko-Mattsson reasons that a standardised mechanism is needed to aid software engineers and managers in adapting their practices that will not interfere with the production and management of software. Also, because the terms are standard, they represent ideas that are common to development and are therefore a fruitful source of abstractions. The trick is to identify the right level of commonality and variability to provide software engineering with processes that can leverage current technologies, trends and techniques [28].

Software projects need to strike a balance between quickly delivering software, satisfying stakeholders, addressing risks and improving ways of working. A software project is faced with two distinct types of complexity, as defined by Senge[14] and Appello [5]. Detail complexity, relating to the complexity and structure of the code, which is typically the focus of developers and analysts and dynamic complexity which relates to the interaction of all the components, including the hardware, software, agents and other dependencies, of a software solution. This dynamic complexity gives rise to emergent behaviours in the project teams and in the actual operation of the software as it provides its functions to the end user. This behaviour can lead to a successful software project or inhibit the delivery and final usefulness of the software.

What is clearly needed is a software engineering theory that embraces the inherent complexity of the software that must be built and complex adaptive nature of the context in which the software development takes place.

Software Engineering Theories and Standards

Software engineering is not short of theories and process models that attempt to solve for the current state of the domain. Each of these brings a perspective on the domain of software engineering but none provide an overall theory or model that caters for both production complexity and the management complexity. Furthermore each perspective provides its own set of definitions to terms that have become over used and over extended. This problem is so prevalent that most software development methodologies now suggest that developers equip themselves with a dictionary of standard terms to make communication among developers easier and more precise [28].

Over the last decade a number of theories and standards have been proposed to define the core elements of software engineering.

Systems Process Engineering Metamodel (SPEM)[29] was adopted by the OMG as a standard for modelling software engineering processes. This standard is a comprehensive model with a very complex language for defining software processes. The standard is designed to be used by method engineers and based on the idea that software processes are software and can be defined as a software program. Consequently, it has a well-defined object oriented metamodel and derived domain model that can be theoretically instantiated as objects in an object oriented language.

Elvesæter et al [30] indicates that SPEM 2.0 does not support enactment of the activities and processes that it defines. These are typically enacted in project management tools as work breakdown structures that are derived from the set of defined activities.

At the core of the SPEM 2.0 specification is the idea that a practitioner performs a set of activities in a set sequence on a set of work products. The work products provide the linking between processes and practitioners and the process ends when a work product has been completed to a defined level of detail[31].

Nardini et al[31] also expresses the concern that the implementation of SPEM in practice is not simple, as the specification is generic and provides no guidance on how to use it. Furthermore it lacks a formal semantic definition for its concepts which will reduce the consistency of any methods developed across organisations.

Kuhrmann et al[32] conducted a study to determine where software process development stands today. They discovered that of the plethora of software development process models that are discussed in literature very few are documented and only two are used in practice. Furthermore, there is no evidence of ISO 24744 or OPEN development outside of the original standards and academic papers referencing ISO 24744 where it supported the instantiation of objects through a discussion of powertypes and situational engineering.

There is evidence of academic papers that have attempted to define a theoretical domain model for software engineering. Kruchten[33] used the Function-Behaviour-Structure framework as a theoretical model and applied it to software engineering processes. This model identifies key engineering design elements. The model was designed to be applicable to any engineering domain. Kruchten noted that software engineering lacks a fundamental theory, making the analysis experimental. The model allowed Kruchten to reason about how to cater for multiple entry points into the software engineering lifecycle where reverse engineering and reusing components is standard practice. The model also illustrates how an underlying theory can be used to influence the kinds of modelling and specifications that can simplify the description process.

Kruchten has more recently proposed a model of software development that abstracts software development into core components that are always evident in any endeavour. He identifies the core concepts of intent, product, work and people and overlays this with project variability components and details how the tool can be used for analysis of a software engineering endeavour and the underlying process models. This model factors in the variable components that are often ignored in more traditional process models and which are the greatest contributors to project failure[34].

Defining Software Development Practices

Henderson-Sellers[35] argues that successful methods that help software engineers build quality software system are those that are constructed from small atomic 'chunks' of method process from a repository of predefined method fragments. These chunks can then be linked together in a method that satisfies a particular situation[35]. These atomic components need a strict definition and an overall metamodel. Henderson proposes a metamodel to describe these chunks that contain a process part and a product part that is influenced by the process part. His model also provides for a description of the usage of the method chunk. The proposal does not provide any guidance on how to take a method and break it up into fragments and chunks, relying on the ISO 24744 standard as the underlying metamodel and the Open Process Framework as the method repository.

Method Languages and Model Comparisons

A review of the literature that researches method languages shows that the available literature is limited to comparisons of the SPEM 2.0 metamodel and the ISO 24744 standard by Henderson-Sellers[36] and this is limited to a discussion of issues with the theoretical object models of the metamodels. Elvesaeter et al[37] compares the Essence model and SPEM 2.0 for the purposes of migrating a SPEM 2.0 model into an Essence model. This paper does not use the Essence as a framework to analyse an existing software engineering model.

The literature review also revealed no theoretical software engineering model used as a framework to assess a model currently used in industry. There are, however a number of papers that compare theoretical models and describe their home grounds [38],[8]. These comparisons are focussed on helping organisations select a model based on the project and organisational characteristics.

The Value of SEMAT

From the literature study it is clear that there are a number of frameworks that have been proposed and can be used to evaluate software engineering models and methods, and only one metamodel that has been used in industry. Dwolatzky[39] states that software engineering is hampered by a split between academic definition and research and what is considered useful in industry. What is needed is a common theory for software engineering that combines academic theory, industry experience which is sometimes considered best practice and proven principles.

Software Engineering Method and Theory (SEMAT) started in 2009 with a call by Jacobson, Meyer, and Soley to *" Refound software engineering based on a solid theory, proven principles and best practices"*[40]. This call was followed by a vision statement [41] and work commenced on developing a kernel of core concepts and defining a theoretical base to underpin software engineering as a discipline. A kernel was then published in response to the Object Management Group's request for proposal: "A Foundation for Agile Creation and Enactment of Software Engineering" in 2012[42].

SEMAT as an initiative has the potential to provide a common theoretical framework that will help academics and industry practitioners debate and agree software engineering best practice and methods. SEMAT provides a set of concepts that are equally useful for a developer team to define their way of working and for the organisation to use the same concepts to understand how they need to set up their project portfolios in a way that best fits the project and organisation characteristics [43].

This will help remove duplication of practices and processes and help change managers introduce a set of consistent models that provide both agile and plan driven approaches. Project managers and

engineers now have a flexible way to align and agree a practical way of working across the multiple systems and teams that are required on any given complex project [40]. Furthermore the Essence kernel makes it explicit that the sociotechnical aspects of software engineering are not second class citizens [9].

The Essence kernel has proved to be practical, providing real world value in developing a framework for growing methodologies in practice and as a lightweight software process improvement approach[44]. The Essence kernel has also been found useful for team reflection and providing a mechanism to help transition projects in a structured way[45].

SEMAT is not without its critics. Concerns have been raised about SEMAT attempting to create a generalised, universal theory that can be applied in any context and that the complex development context can somehow fit into a standard, predefined set of alphas [46]. Cockburn [47] has challenged the underlying reasons that support the original call to action and claim that they are inaccurate, misleading and not based on any agreed evidence. He ultimately questions the validity of developing a theory of software engineering when, as he maintains, it is still not understood what happens on most software projects.

The current work in SEMAT and on the Essence kernel is focussed on using the kernel as a framework to improve software engineering tuition[45] and as a consulting tool for improving software processes[44].

Chapter 3 Research Objectives

The objective of this study is to evaluate the usefulness of SEMAT (Software Engineering Methods and Theory) as a framework for software engineering process analysis. To achieve this objective, the SEMAT[9] model will be evaluated against an industry lifecycle to gain a deeper understanding of the Essence kernel and identify gaps in the model. The evaluation and its resultant analysis model could assist organisations to analyse and recommend improvements to their software system delivery capabilities.

The Research Question

The following research question will guide the development of this objective:

In what way can the underlying model of the organisation's software development lifecycle be critically evaluated by mapping the software development lifecycle onto the SEMAT kernel and does the mapping provide insight into the gaps identified?

To answer this question a set of hypothesis are formulated to help narrow down the scope of the investigation. The following hypotheses have been identified:

1. The SEMAT framework can be mapped to a custom industry lifecycle and it can be used to describe how the entire process hangs together
2. The SEMAT framework is a useful construct in analysing software engineering models and is useful in identifying gaps and guiding the selection of appropriate practices and methods to address these gaps

The Research Approach

The selected approach is designed to facilitate an understanding of the context of the organisation and to gain an understanding of the existing software development cycle. This understanding will provide a foundation to illustrate how the existing lifecycle and its related practices can be mapped to the SEMAT kernel using the kernel and its language. The resulting mapping will be used to describe how the SEMAT kernel and language can be used to analyse the practices and processes contained within the organisation's repository. The analysis will provide a set of suggestions for consideration as feedback to the SEMAT team and a framework for analysing the organisation's processes and practices.

The steps of the methodology are outlined here:

1. A literature survey of SEMAT and other frameworks that are useful in analysing software engineering processes and practices
2. Researching and reviewing relevant literature
3. Researching and understanding the SEMAT concept and identifying key concepts
4. Researching the organisation's processes and practices and identifying its underlying language and concepts
5. Associating key concepts of the organisation's method to the SEMAT Kernel and language
6. Using the mapped framework to identify gaps and possible mechanisms to resolve these gaps in both the organisation's lifecycle and the SEMAT Kernel
7. Confirming the hypothesis and subsequent answering the research question

The Analysis Framework

A framework for the analysis of a software engineering lifecycle was developed to help with the investigation and analysis. The framework helps identify the underlying domain model, the key elements of the model and how these elements fit together in practices. This framework will provide a foundation to develop a generic analysis model that could be used to assess any organisation's software engineering capability.

The framework consists of a set of questions that are designed to help analyse an existing set of practices and methods in terms of the language of the SEMAT kernel. The set of questions used as the basis of this research is described here:

- What are the essential concepts that practitioners work with (alphas)?
    - Do these essential concepts have states?
    - Do checklists exist to determine the states?
    - What lifecycle concepts can be expressed in terms of the states of a collection of alphas?
    - What collection of work products manifests an essential concept?
- What are the essential activities and tasks that are executed?
    - Which concepts do they drive?
    - Which states do they drive?
    - What criteria are to be fulfilled to measure the completeness of an activity?
- What work products provide evidence of the essential concepts?
    - What activities provide guidance on creating work products?
- What patterns are defined? (phases, milestones, gates)
- What practices can be identified?

The Case Selection

The organisation selected is one of the banks in South Africa with a significant investment in technology and software intensive systems. In recent years the industry has had to deal with the rapid pace of technological change, customers increased usage of mobile technology and an increased use of technology solutions to gain competitive advantage. This market-driven change has challenged the organisation's software development capability resulting in a desire to improve their existing software and systems development processes. The company recently embarked on a journey to introduce a new system development lifecycle to improve its software delivery capability. This new methodology promises to help the organisation reduce the time and cost it currently takes to introduce new software into the organisation and to reduce the risk of introducing this software and its related infrastructure components into the operational environment. The new lifecycle is an ideal candidate to use as a case for assessing the new SEMAT standard, as it is formally defined and documented in a tool and easily accessible to the researcher.

In the research described here, the unit of analysis was at the level of the described engineering lifecycle and its template artefacts. The study did not consider any projects that used the method, nor did the study consider any artefacts delivered as part of any endeavour.

The Data Collection

A number of data sources were used as sources in this study. The primary source was the internally published method website, which acted as the organisation's method repository and contained all

the processes, activities, work breakdown structures, templates and techniques. The repository is a live repository and as such the lifecycle is under continuous modification for improvement and the research standardised on version 9.0 of the method for the purposes of analysis. The organisation's published method, and related documentation is proprietary, and due to license constraints is not in the public domain and consequently cannot be distributed to external parties.

The researcher used two other sources of information to supplement this primary information, an extract of the method activities for use in method adoption workshops and the Technical Review Board workbook which is used extensively in the governance model of the new lifecycle. The research method required the use of the published SEMAT Essence specification, version 1.0, published on the OMG website [9].

Chapter 4 Research Theory and Data

This chapter details the research work that was completed as part of the critical analysis of the Essence model and the Bank's lifecycle. The chapter is structured into three sections, each describing an aspect of the research work. The first section introduces the Essence model and discusses some key aspects of the standard to facilitate an understanding of the terms and concepts used in later sections. The second section introduces the Bank's system development lifecycle in the language of the Essence standard. The final section discusses the research work completed to critically analyse the Essence standard against the Bank's lifecycle.

SEMAT Essence Overview

SEMAT is a software engineering community initiative designed to address many of the current issues in software engineering [40]. A key outcome of this community is a specification called the Essence that is intuitive and provides a simple graphical syntax to simply capture and explain existing practices [15]. The overarching vision is to create a specification that supports enactment of software engineering by practitioners much like the Agile Manifesto supports more agile software development practices [30].

The Essence, as an emerging standard, is composed of a kernel of essential elements of a software endeavour and a language to describe these elements. The standard enables the essential things to work with, essential things to do and essential things to know to be defined and described in a consistent, coherent and easily understood language with both a textual and graphical syntax [9]. A feature of the Essence is the ability to separate the elements that define the domain of software engineering from how it is used in a specific software endeavour or organisational context [15].

The domain of software engineering is captured in a kernel and described using a small set of language concepts. The kernel describes the core concepts that can be used to describe any software engineering endeavour. The kernel is organised into the three areas of concern of Customer, Solution and Endeavour and composed of a set of:

- Alphas, which are an abstract representation of the core things to work with
- Activity spaces, which are containers for the core things to do
- Competencies, which are the core things to know [9]

The kernel provides a framework on which a set of practices can be composed. A practice will extend the kernel alphas with a set of concrete work products and extend the activity spaces with concrete activities. This model is illustrated in the following diagram, Figure 2, adapted from the Essence – Kernel and Language for Software Engineering Methods [9] which provides a visual representation of the core components of the kernel, their conceptual relationships and highlights the separation of the kernel and the practice elements that are used to enact the method [9].

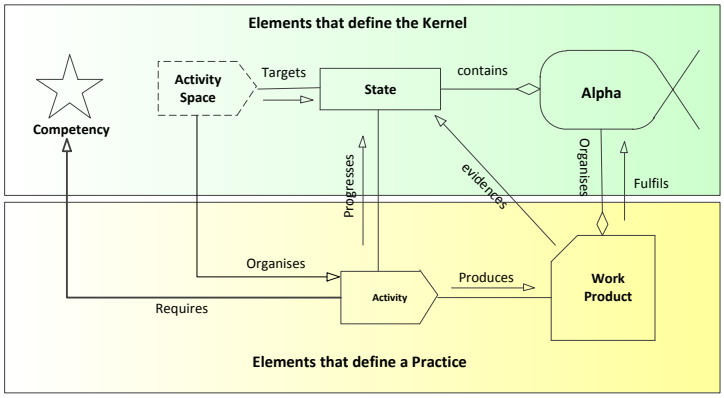


Figure 2: Core components of the Essence (adapted from [9])

The Alphas

The alphas, as defined in the Essence, are the core elements in the kernel. They represent:

- The key concepts of the things to work with in software engineering
- A mechanism to assess the state and health of any software engineering endeavour
- A standard definition of components that can be used to describe any software engineering method or practice

Alphas are not a simple functional decomposition of an endeavour into core work products, they are abstract concepts that represent elements that are the most important to understand, monitor and progress. Alphas are common concepts that are present in any method or practice and do not describe any method or artefact. This abstract definition allows for a domain model to be developed that details how these elements are related without prescribing a particular implementation. This enables the alphas to describe both an agile method and a traditional waterfall method with the same well-defined constructs. This ability to describe all methods and practices in the same way leads to a constructive method of comparison. Figure 3, borrowed from [9], illustrates the Essence kernel alphas and how they are grouped into their areas of concern and details their relationships with each other [9].

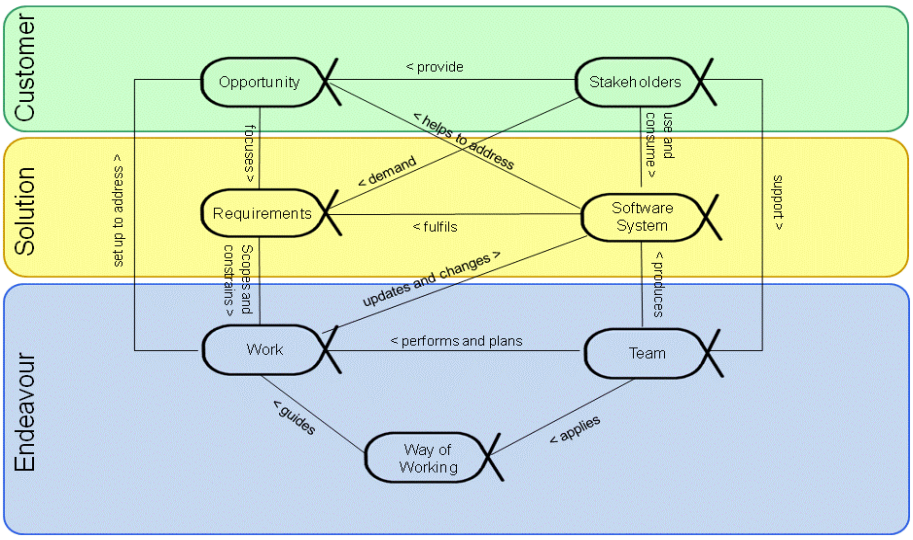


Figure 3: Essence kernel alphas [9]

Each alpha has a set of states that support and guide the progress of an endeavour and attached to each state is a well-defined checklist that helps a team identify that a state has been achieved. For example, the *Opportunity* dimension of an endeavour will move through the following states as the endeavour progresses[9]:

- *Identified*
- *Solution needed*
- *Value established*
- *Viable*
- *Addressed*
- *Benefit accrued*

The team can assess the state of their opportunity by applying the checklists associated with each state to their current situation. This approach helps the team plan the activities required to progress this dimension of their endeavour to the next state [15].

An important aspect of the alphas and their relationships is the dynamic multi-dimensional view of the state of all aspects of the endeavour that is provided [9].

The alphas not only provide states, but provide a framework to organise and build more detailed views. Alphas can be decomposed into sub-alphas to allow for more detailed, practice specific states and work products to be described. These sub-alphas provide more granular mechanisms to progress the state of the superordinate alpha in some measurable way. Figure 4, based on an example diagram in [9], illustrates an example of this decomposition, using elements from the Bank's methodology.

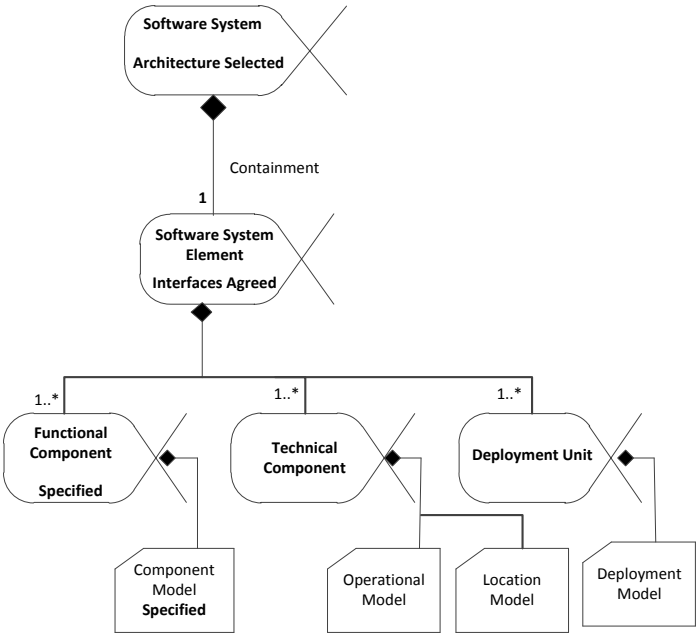


Figure 4: Alpha decomposition (adapted from [9])

The Activity Spaces

The Essence provides a set of activity spaces that represent abstract things to be done. They support the alphas by providing an abstract view of the type of activities that are required to progress the

alphas states. Each activity space is a placeholder for a collection of activities, actions or tasks but is not a process view, as they are a "*network of collaborating elements*" [15] with no inherent sequencing defined. These activity spaces drive completion criteria that map directly to the states of the alphas that they address. An activity space may impact one or more alphas and the activity space completion criteria represent specific states of these alphas. This provides a multi-dimensional snapshot of the state of the important elements of the endeavour at the completion of a set of activities. This snapshot is not based on the completion of a set of work products, but rather on the attainment of criteria that that will provide project managers and teams a complete view of the state of a project.

The activity spaces are organised into three areas of concern, but unlike the alphas are not directly related to each other. Their relationships are mapped via their alphas and the alpha relationships. Figure 5, borrowed from [9], illustrates the core set of activity spaces defined by the Essence [9].

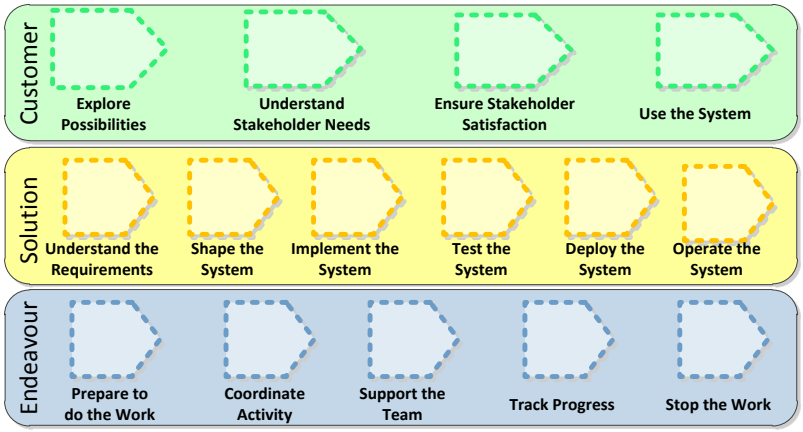


Figure 5: Essence activity spaces [9]

Each area of concern contains a set of activity spaces that progress the alphas in that area of concern. The activity spaces take alphas as input and after some work is performed the alphas are potentially updated and their states may have changed. For example, the activity space *Explore Possibilities* has no alphas as input but after possibilities have been explored via a set of practice activities and tasks, the *Stakeholder* alpha state should have progressed to *recognised* [9]. If the alpha state has not been achieved after assessing it using its checklist then the activity space completion criteria has not been met, and a decision can be taken on what work must be completed to progress the alpha to the correct state.

Activities that provide guidance on how to progress a specific alpha state are placed within the activity space whose completion criteria address that alpha state. This construction provides a mechanism to ensure that an activity space contains a coherent, non-redundant and complete set of activities when practices are defined and a method is assembled. Activities, in addition to providing specific technical guidance on the completion of work, may indicate the competency required for performing this work. This facilitates the creation of practical and useable methods and practices [15]. Figure 6, adapted from [9] and [48], illustrates how all the Essence elements are related.

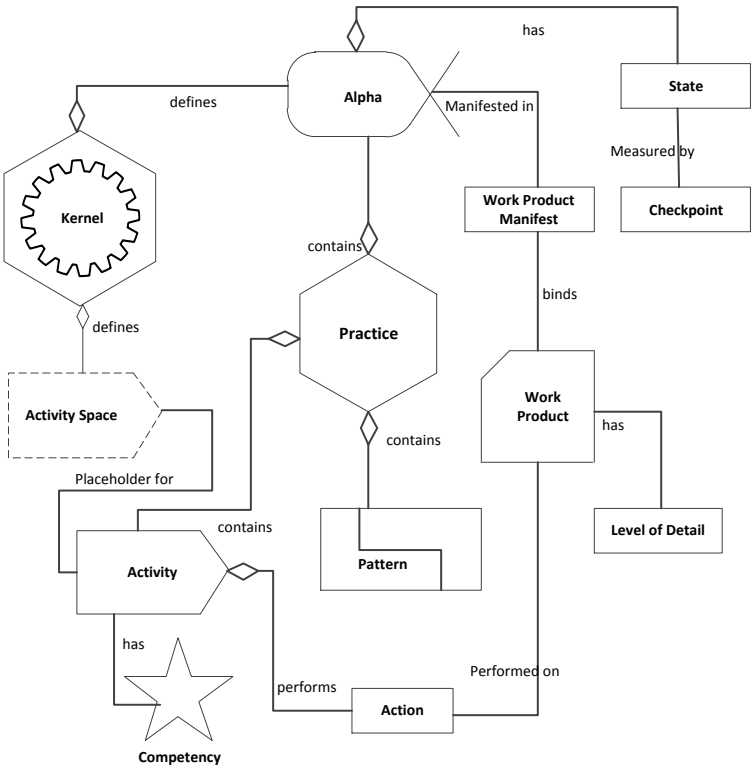


Figure 6: Essence components of a practice (adapted from [9,43,48])

The Practices and Patterns

A practice is the Essence's mechanism of providing a way to define a chunk of the work to be done and the processes required to sequence this work. A practice defines the concrete work products, activities and patterns required to guide and help practitioners achieve their purpose [9]. A practice can be considered a self-contained chunk containing the minimum components required to achieve a purpose [35]. These chunks can then be assembled into a method. Approaching method creation and adoption in this way allows the method to be developed from actual artefacts and process used by practitioners and allows these practices to be easily discussed and reused [44]. Figure 6 illustrates how a practice links into the Essence kernel and uses the kernel components.

The Essence make use of the concept of a pattern to cater for concepts that do not fit into the basic elements and are not things that can be worked with, cannot be allocated as tasks nor are they competence related. Typical lifecycle elements that fit this definition are phases and roles, which are there to add structure and groupings[9]. Figure 7, based on [9] illustrates a typical phase pattern found in a bank development lifecycle. The work is partitioned into a phase called *solution outline* that delivers the *Business Model* and *Architecture Model* work products.

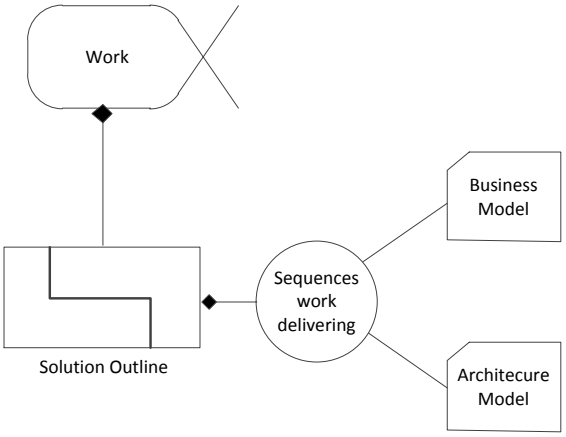


Figure 7: Phase pattern (adapted from [9])

The Bank's Model

The Bank's model is a general purpose lifecycle whose overarching mandate is to manage and balance the process of introducing change into the already operational banking system landscape without introducing instability and failures. This mandate of managing risk drives the core features of the lifecycle.

The Bank's systems development lifecycle (SDLC) is constructed of three elements:

- The underlying metamodel for documenting and publishing the lifecycle
- The method model, including the content and the underlying practices and techniques
- The process and management model

The illustration in Figure 8, describes the elements in Bank's model:

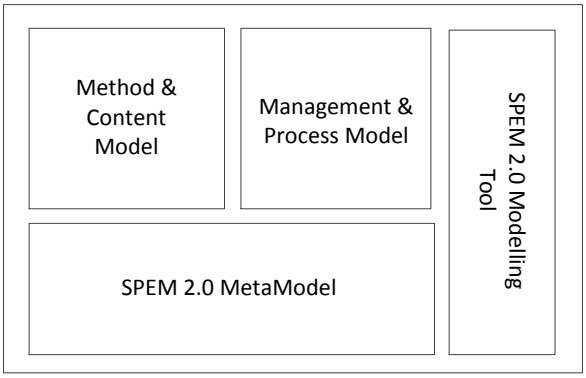


Figure 8: The Bank's SDLC elements

The Meta Model

The core method of the Bank is documented and managed using a framework and tool that conforms to the Software and Systems Process Engineering Metamodel (SPEM 2.0) [29]. Figure 9, adapted from [37], [49] and [50] , illustrates the key language concepts that make up the Bank's lifecycle model.

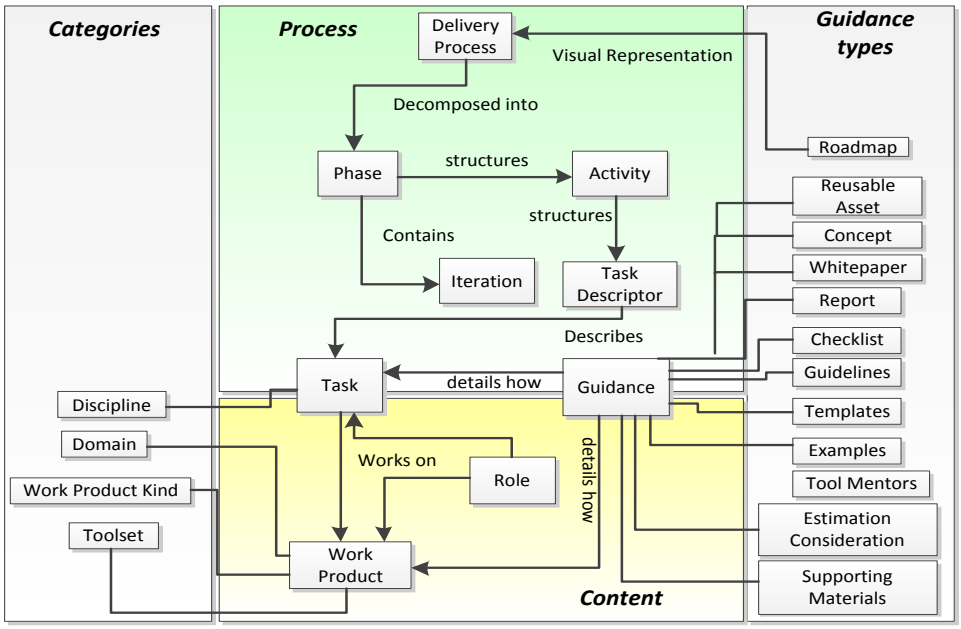


Figure 9: Key SDLC language concepts (adapted from[37,49,50])

The metamodel contains four key areas of concern:

- The process area
- The content area
- The categories area
- The guidance types area

The process area of concern contains elements for structuring and sequencing activities and tasks. The process area provides mechanisms for describing what has to be done and when it should be done including dependencies on other activities. It does not provide any mechanism to describe how the work is to be completed. The content area of concern provides the elements needed to define how a task is to be completed [49]. This definition includes work products and guidance types that describe in detail how to do the work. The task element links the process and content models.

An important principle that is evident in the use of the model, and can cause confusion to practitioners, is the idea of reusable content and processes. Each element in the model is considered for reusable content and is defined once and then the same definition is reused across the lifecycle [49]. An example illustrating this principle is the task *Detail Key System Use Cases*; it is a task that is defined once with all the work products, roles and guidance associated with the task. The task is then linked to multiple activities in various phases, resulting in a single description of *Detail Key System Use Cases* listed in multiple phases.

The model allows for a task to be categorised into disciplines such as *test management* which categorises all tasks related to testing. This assists a practitioner in identifying a list of tasks required for their job type. A discipline is not equivalent to a practice; a practice is a type of guidance which creates a coherent composition of tasks, roles and work products to fulfil a piece of work[49]. A discipline is a list of tasks that can be considered part of a domain; it provides no structure or guidance for completing any work.

The Method Model

The metamodel described previously provides a taxonomy and structure to capture a breakdown of the work to be completed. The tasks and guidance in this breakdown provide details of how to complete the various work products, but the metamodel does not describe why the tasks and work products are required and how they fit together to define a working software system. This information is captured in the guidance types and provides the model from which the tasks, work products and roles are derived. Figure 10 provides a conceptual view of the underlying engineering method and is built up from the work product model in [50] and [51]. It highlights the core elements of the method and how they are sequenced to provide a complete solution to the business concept. It illustrates an understanding that the business concept that drives and informs the requirements is not completed upfront, but is developed over the life of the endeavour. This will impact the stability of the requirements and the engineering approach required to develop software to fulfil the requirements and satisfy the business concept.

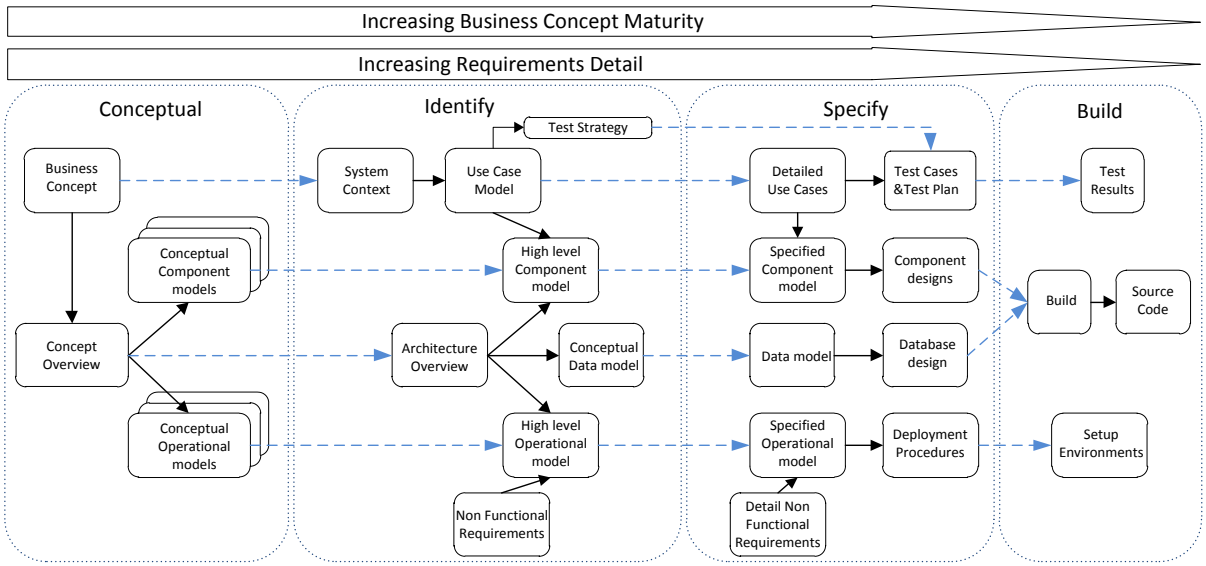


Figure 10: Method metamodel

The overarching engineering objective is to ensure alignment to the emerging business concept and integration into the existing operational environment. This objective is achieved by moving the solution through:

- A conceptual phase, designed to explore possible solutions to the business concept
- An identify phase, designed to identify use cases, components and data entities and pull together an architecture overview of the solution
- A specify phase, designed to detail the use cases, components, data models and facilitate the design of the components, databases and test cases. The application deployment is modelled and deployment procedures developed to ensure the solution can be operationalised
- A build phase, where the software is developed, built and tested
- A deploy phase (not illustrated) [50]

It must be noted that the model allows for both a waterfall and an agile incremental development approach, with *specify*, *build* and *deploy* phases part of an iterative model.

The Management and Governance Model

The method and its related activities and tasks are organised into a structure that provides a means to manage the project and apply the appropriate governance. This structure ensures that the organisations objectives are achieved through appropriate monitoring and risk management practices.

The management model is structured on the Project Management Institutes project management process and contains seven phases [50]. Figure 11 illustrates how the model is structured and is based on a diagram obtained from the Bank's method [50,52].

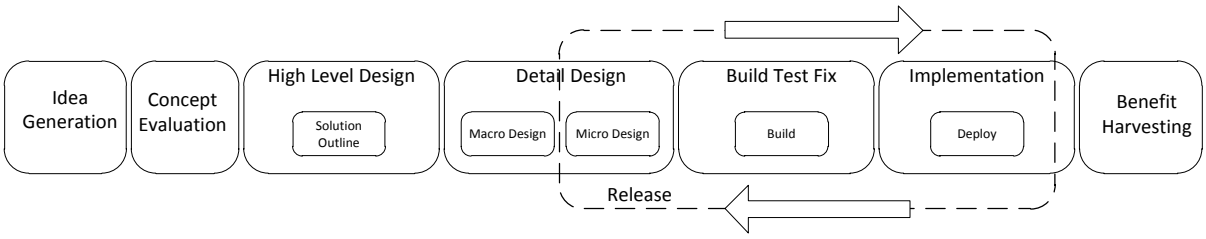


Figure 11: Phases of the management model

The Seven phases illustrated in the Figure 11 are briefly described:

- *Idea generation* phase which is the entry point of an initiative into the portfolio planning processes.
- *Concept evaluation* phase where the project is initiated and teams and scope are identified
- *High level design* phase where the solution is outlined and the project is formally planned and project execution funding is allocated
- *Detail design* phase where the approved solution is analysed, defined and designed
- *Build, test, fix* phase where the software is developed, tested and bugs are fixed
- *Implementation* phase where the solution is deployed into the operational environment
- *Benefit harvesting* phase where the business case benefits are measured in the production environment to verify the actual benefits achieved[50]

Each phase is further decomposed into one or more technical solution phases, which contain the activities and tasks directly related to the software and solution development. These phases are:

- *Solution outline* which contain the tasks for defining the solution approach and architecture
- *Macro design* which contain the detailed solution tasks
- *Micro design* which contain the component software design tasks
- *Build,* which contains the tasks for coding and testing
- *Deploy* which contains the tasks for deploying the solution into production[50]

The governance model is centred on governing three core elements of the endeavour[52,53]:

- The funding of the work
- The project management activities
- The software development and solution activities

For the purposes of this research the focus will be applied to the software development and solution governance activities, the funding and project management governance activities will not be

considered. Figure 12 illustrates a consolidated view of the governance and management model and is built from information gathered from the Bank's method [50,52,54,55]. It illustrates the key governance check points and how they are related to the model's phases.

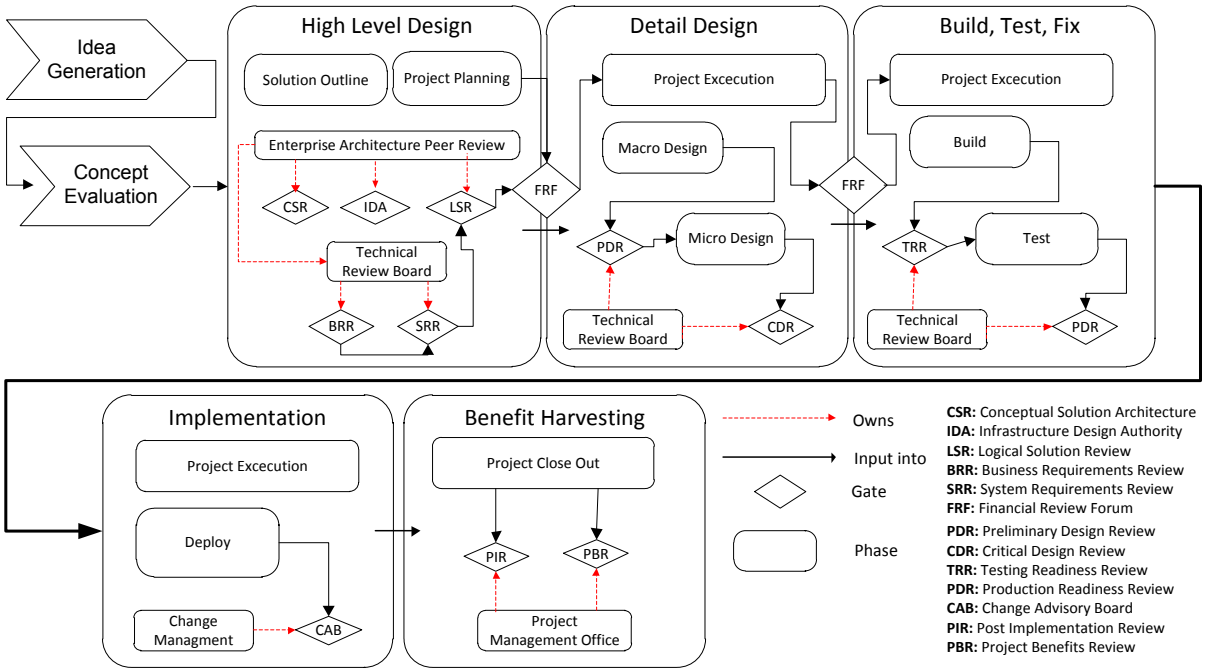


Figure 12: Governance model

The governance of the solution is owned and managed by the enterprise architects and is implemented through a set of reviews and boards mandated to assess and manage specific domains:

- The enterprise architecture peer review (EAPR)
- The infrastructure design authority (IDA)
- The technical review board (TRB)

The EAPR is the overarching board that delegates project level governance to virtual technical review boards, which are setup and managed within the individual projects. The EAPR reviews the conceptual solutions and runs the infrastructure design authority which is mandated to review the infrastructure components of a solution and to manage the risk of introducing new components into the operational environment[53,54].

The TRB governance gates perform a number of functions that support the endeavour. Each gate is constructed of a set of work product slots, a set of continuous assessment criteria, a review log and a set of scores. These elements together create the capability to assess a software endeavour at a point in time and create a baseline. The primary mandate of each gate is to build shared accountability and to ensure alignment and quality for the solution[52,55,56]. Each gate has a specific alignment and quality focus.

There are six TRB gates defined[52,56]:

- Business requirements review (BRR), which creates a customer baseline
- System requirements review (SRR), which creates a system baseline
- Preliminary design review (PDR), which creates a component baseline

- Critical design review (CDR), which creates a design baseline
- Testing readiness review (TRR), which creates a test baseline
- Production readiness review (PRR), which creates a production baseline

Identifying the Bank's Alphas

The foundation of the Essence kernel is the concept of an alpha, which is an essential thing that engineers will work with in building software intensive systems and this was the starting point of the research. The Bank's existing lifecycle documentation was unpacked to identify concepts that are considered important. The approach considered the governance model, the list of activity definitions, the work product descriptions and linked technique papers in identifying potential alphas. The list of activities proved too granular to effectively identify these alphas and was not used directly in the identification process; however the activity descriptions related to an identified alpha were used to help understand how the alpha fitted into the whole process.

The governance model provided a rich source of activities, work products and potential alphas that are considered important enough to formally review and track. The governance model makes extensive use of a concept called a slot. These slots represent the essential things to work with in the Bank's lifecycle and provided an initial list of potential alphas. Each governance gate review assesses a set of slots and has a defined set of criteria for each type of review. Furthermore each slot has a defined set of work products that fulfil the slot's requirements depending on where in the lifecycle the slot is being referenced.

Each review is designed to measure the technical health and progress of the work products and thus the state of a slot at specific points in the lifecycle. Figure 13: Conceptual model of a slot, illustrates the concept of a slot defined using the Essence language and based on a diagram from [9].

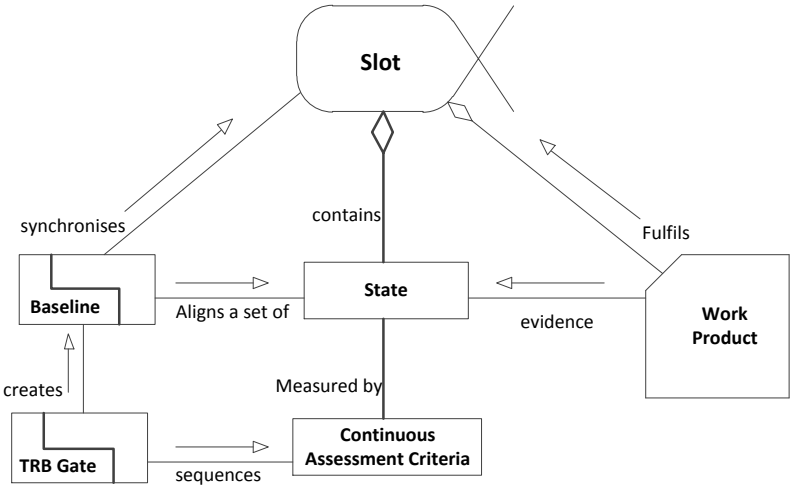


Figure 13: Conceptual model of a slot (adapted from [9]**)**

The Bank's lifecycle defines thirteen slots that represent placeholders for concrete work products. These slots are defined once and reused across the phases of the lifecycle and are fulfilled by the same work products at different levels of detail and states of completion. The table in Table 1 lists the slots and links them to their corresponding gates, refer to Appendix D: Mapping CAR's to Slots for further details. The table shows that not all slots are reviewed by a specific gate and the *Project Work* slot is not reviewed by any gate.

Method Slots	Technical Review Board Gates					
	BRR	SRR	PDR	CDR	TRR	PRR
Business Requirements	Y	N	N	N	N	N
Existing Environment	Y	Y	Y	Y	N	N
Business Model	Y	Y	Y	Y	N	N
Application Model	N	Y	Y	Y	N	N
Requirements Report	N	Y	Y	Y	N	N
Test Approach	N	Y	N	N	N	N
Architecture Model	N	Y	Y	Y	Y	Y
Standards & Procedures	N	Y	Y	Y	Y	Y
Test Pack	N	Y	Y	Y	Y	Y
User Experience Model	N	Y	Y	Y	N	Y
Technical Implementation	N	N	N	N	N	Y
Project Work	N	N	N	N	N	N
Project Definition & Scope	Y	N	N	N	N	Y

Table 1: Slots linked to gates

Reviewing the list of work products created and used in the lifecycle revealed a further set of alphas that provide a deeper level of granularity. This set of alphas is evidenced in the work products that are linked to the slots, but are not necessarily reviewed by a gate. The *Deployment Unit* is an example of an alpha that is used in developing the *Node* and consequently the *Operational Model*, but is not reviewed as a standalone work product. Table 2 lists a subset of this list of Bank alphas and the work products that realise them. A more detailed table is provided in Appendix B: Bank's Alpha Table

Identified Bank Alpha	Work Products
Component (Functional Component)	· Architecture Overview · Component Model · Conceptual Solution Architecture · Component Design · Component Specification · Source Files
Node (Operational Component)	· Node Model · Node · Connection · Operational Model · Zone Model · Location Model
Deployment Unit	· Operational Model · Node Model
Data	· Data Migrated · Data Migration Specification · Data Model · Database Transaction Descriptions · Metadata Strategy · Physical Database Design

Table 2: Bank alphas identified in work products

Identifying the Bank Alpha States

Each governance gate reviews a collection of slots that is specific to each gate. The work products that make up these slots are reviewed against a set of *continuous assessment requirements* (CAR) questions. These CAR questions indirectly measure the state of the slots. Table 3 illustrates the mapping of the BRR gate, the slots that are reviewed and the states derived from the gate's CARs. This mapping reveals that the slots have a set of states, and it is discovered that the gate baselines are expressed in terms of the states of a collection of slots, and by reference a collection of Essence alphas. Thus a *Customer Baseline,* which is a milestone set by the BRR, can be considered achieved if

the business requirements are *defined*, *traceable*, *agreed* and have a *priority*; the as-is architecture is *validated;* the as-is business process is *identified* and *validated;* any *gaps identified* and the to-be business processes have been *defined*. The data in Table 3 reveals that a slot may be constructed from more than one bank alpha as is evident in the *Existing Environment* slot where the slot contains work products and states for both business process and architecture. The definition of a slot does not prevent this scenario and may be an indicator that a slot is not a good match to the Essence alpha definition.

Slot	Gate	Baseline	Slot State
Business Requirements	BRR	Customer	· Business Requirements Defined · Business Requirements Traceable to Scope · Business Requirements Agreed · Business Requiremenst Priority Agreed
Existing Environment	BRR	Customer	· As-Is Architecture Validated · As-Is Business Process Identified · As-Is Business Process Validated · Gaps Identified
Business Model	BRR	Customer	To-Be Business Process Defined

Table 3: Slot states for a gate

This same process was employed to identify the states that are linked to the Bank alphas, listed in Table 2, that were previously identified through the detailed review of the Bank's work products. This work proved difficult as the states of these alphas were not always explicit, used or referenced directly in any activity or review and thus required a detailed review of the technique papers, templates and tasks related to the alpha's work products. Figure 14 illustrates this complexity and the method relationships.

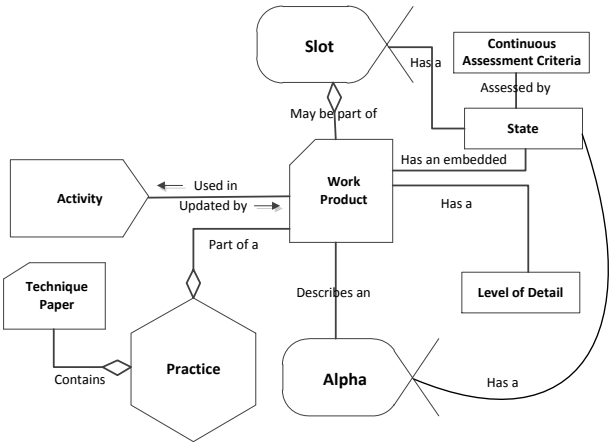


Figure 14: Information contained in a work product (adapted from [48]**)**

The Bank's lifecycle contains a method that is designed around functional, operational and requirements domains. The functional domain contains work products and activities concerned with the functionality of collaborating software components of a software system. The functional aspect is expressed as one or more models that represent the static structure and dynamic behaviour of the components in the system. The functional domain progresses a *Functional Component* alpha.

The operational domain comprises work products that are concerned with the distribution of system components across the organization's geography in order to achieve the required service level characteristics (performance, availability, and so forth). The operational work products are normally represented by one or more operational models that describe the type and location of hardware nodes, connections, network topology, in terms of the placement of deployment units.

They are also concerned with the systems management functions and activities needed to maintain the system components such as software distribution and responding to alerts. The operational domain progresses an *Operational Component* alpha.

The requirements domain is decomposed into business requirements and system requirements and the testing of the requirements. The business requirements sub-domain comprises work products and activities to define the business and help to develop the business opportunity. The work products are also used to document the business processes, the business direction and business cases. The business requirements domain progresses the *Business Context*, *Business Process* and *Opportunity* alphas

The system requirements sub-domain addresses the development, documentation, and management of the requirements for the software, hardware or system developed. The work products and activities are concerned with use cases, non-functional requirements and actors. The system requirements domain progresses the *System Requirements* alpha

The testing sub-domain defines the work products and activities that address the strategy, planning, managing and executing activities of the testing process. The sub-domain also defines activities and work products for analysing test results, and reporting on testing activities. The testing sub-domain progresses the *Test* alpha which is a sub-alpha of the *System Requirements* alpha. The data in Table 4 represents a subset of the outcome of this work. A complete table is provided in Appendix B: Bank's Alpha Table.

Bank Alpha	States	Level of Detail	Work Products
Component (Functional Component)	· Responsibilities Identified · Responsibilities Allocated · Component Specified · Component Developed · Component Deployed	· Logical Application · Specified/Logical · Physical	· Architecture Overview · Component Model · Conceptual Solution Architecture · Component Design · Component Specification · Source Files
Node (Operational Component)	· Node Identified · Components Allocated · Connections & Interactions Specified · Physical Node Sized · Node Configured	· Logical Application · Logical · Physical · Unsized · Ranged · Sized	· Node Model · Node · Connection · Operational Model · Zone Model · Location Model
Deployment Unit	· Identified · Placed · Connected · Tested	· Logical Application · Logical · Physical	· Operational Model · Node Model
Data	· Conceived · Resolved · Normalised · Transformed · Implemented	· Conceptual · Logical · Physical	· Data Migrated · Data Migration Specification · Data Model · Database Transaction Descriptions · Metadata Strategy · Physical Database Design

Table 4: States for non-slot alphas

Mapping the Bank's Alphas to the Essence Alphas

Once the list of alphas was defined and their states identified, these alphas were mapped to the Essence alphas. Each slot, as a potential alpha, was assessed to see if it could be mapped to at least one of the Essence kernel alphas to identify gaps in the Bank's lifecycle and assess the validity of the set of kernel alphas. The Essence alphas were mapped to both the slots via analysis of the slot descriptions, the descriptions of the work products that fulfilled the slot and the activities that create and progress the work products, and the alphas identified from reviewing work products, technique papers and activities not specifically linked to the list of slots.

Each slot's work products were assessed to see if they described a kernel alpha in some way. If a work product could be linked to a kernel alpha in some way, either by the work product description or by its related task descriptions it was indicated in a matrix as a 'Y'. If there was no link the cell was left blank. This analysis was captured in a matrix, mapping the slots against the Essence kernel alphas. It is evident from Table 5 that each slot is mapped to at least one Essence kernel alpha, refer to Appendix C: Mapping Slots to Baselines for more details. It is evident from the matrix that the Team alpha is not fulfilled by any slot work products and therefore not included in any slot. This matrix does not capture any information regarding the state of the Essence kernel alphas.

Method Slots	**Essence Alphas**						
Table derived from information contained in work products linked to the slots.	***Opportunity***	***Stakeholders***	***Requirements***	***Software System***	***Team***	***Work***	***Way of Working***
Business Requirements	Y						
Existing Environment	Y	Y					
Business Model	Y						
Application Model				Y			
Requirements Report			Y				
Test Approach						Y	
Architecture Model				Y			
Standards & Procedures			Y	Y		Y	Y
Test Pack				Y			
User Experience Model			Y	Y			
Technical Implementation				Y			
Project Work						Y	
Project Definition & Scope						Y	

Table 5: Slots mapped to Essence alphas

Extending the mapping to the alphas identified through the review of work products, activities and technique papers produced a matrix, a subset of which is illustrated in Table 6. The matrix links the Bank alphas to one or more Essence kernel alphas via the descriptions of the activities and work products that evidence the Bank alpha. The Bank alpha's activities and work product descriptions were assessed against the checklists linked to the Essence kernel alphas to determine if the Essence alpha states were progressed by any of the Bank's work products and activities. A complete table is provided in Appendix B: Bank's Alpha Table.

Bank Alpha	**Activities**	**Work Products**	**Essence Alpha**
Component (Functional Component)	· Outline Conceptual Technical Solution · Outline Technical Solution · Develop Technical Architecture · Develop Physical Architecture · Refine Technical Architecture · Develop Detail Component Design · Develop Solution Software Components · Update Technical Design · Deploy to ETE, QA · Deploy Production Environment · Cutover to Production	· Architecture Overview · Component Model · Conceptual Solution Architecture · Component Design · Component Specification · Source Files	Software System
Node (Operational Component)	· Outline Conceptual Technical Solution · Outline Technical Solution · Develop Technical Architecture · Develop Physical Architecture · Refine Technical Architecture · Deploy to ETE, QA Environment · Deploy Production environment	· Node Model · Node · Connection · Operational Model · Zone Model · Location Model	Software System

Bank Alpha	Activities	Work Products	Essence Alpha
	· Cutover to Production		
Deployment Unit	No activities specific for deployment unit, embedded in guidences and templates	· Operational Model · Node Model	Software System
Data	· Develop Data Design · Refine Data Design · Develop Solution Data Components · Deploy ETE, QA Environments · Deploy Production Environment	· Data Migrated · Data Migration Specification · Data Model · Database Transaction Descriptions · Metadata Strategy · Physical Database Design	
Business Process	· Capture Existing Environment · Outline Business Solution · Develop Business Solution · Refine Business Solution	· Business Event List · Business Rules · Procedures Document · Process Assessment & Analysis · Process Definition · Process Identification · High Level Gap Analysis · Business Function/ System Matrix	Requirements Opportunity
Stakeholders	Obtain Current Organisation Description	· Stakeholder Analysis · Stakeholder Matrix · User Profiles	Stakeholder

Table 6: Alphas mapped to non-slot alphas

Mapping the Essence States onto the Bank's Lifecycle

The Bank's process model tracks six essential milestones called baselines. These baselines form the key indicators of the progress of an endeavour. Each activity and work product is grouped and structured to progress the solution towards these baselines and the governance gates assess the achievement of these baselines. Mapping the Essence kernel alpha states against the baselines provides insight into the validity of the states identified by the Essence model and highlights gaps in the Essence model and in the Bank's model. The table below,Table 7, details the initial mapping of the Essence kernel alphas against the baselines based on the required work product inputs into the baseline and the documented baseline definition.

Essence Alpha	SDLC Baselines						
	Customer	**System**	**Logical Solution**	**Component**	**Design**	**Test**	**Production**
Opportunity	· Solution Needed	· Value Established · Viable	Viable	-	-	-	Addressed
Stakeholder	· Recognised · Represented · Involved	In Agreement	In Agreement	-	-	-	· Satisfied for Deployment
Requirement	Bounded	Coherent	Coherent	Acceptable		Addressed	Fulfilled
Requirement Item (sub alpha)	Identified	Described	Described	-	-	Implemented	Verified
Software System	-	· Architecture Selected	· Architecture Selected	-	-	-	· Demonstrable · Useable · Ready
· Software System Element (sub alpha)	-	Identified	Identified	Interfaces Agreed	-	Developed	Ready
Work	Initiated (Pre-execution Tailored)	Under Control (Execution Tailored)	Under Control (Execution Tailored)	Under Control (Execution Tailored)	Under Control (Execution Tailored)	Under Control (Execution Tailored)	Concluded
Way of Working	-	-	-	-	-	-	-
Team	Seeded	Formed	Formed	· Collaborating · Performing	· Collaborating · Performing	· Collaborating · Performing	· Collaborating · Performing · Adjourned

Table 7: Essence alpha states mapped to baselines

The table highlights some key gaps in the mapping. It is immediately obvious from this mapping that each baseline can be described by a set of Essence alpha states, much like the link between the Essence activity space completion criteria and a set of Essence alphas states. A detailed analysis of this table is provided in Appendix A4: Analysis of Baselines.

The Bank's lifecycle defines a number of phases based on a traditional phase model and sub-phases that provide a structure for the engineering activities. These groupings have activities and work products as outputs that progress the engineering activities and fit into the project management phases of the lifecycle. These engineering sub-phases are:

- *Solution outline* (SO), located within *high level design* (HLD)
- *Macro design* (MD), located within *detail design* (DD)
- *Micro design* (UD), located within *detail design* (DD)
- *Build, Test, Fix* (BTF)
- *Implementation* (IMPL).

Each of these phases moves an endeavour through various levels of detail and completeness. The lifecycle can be described in terms of the Essence alpha states by using actual activities described in the lifecycle and in which phase they are executed.

This mapping provided the following Table 8 that captures the full lifecycle, the phases and the Essence alpha states.

Essence Alpha	Phase					
	CE	HLD	DD		BTF	IMPL
		SO	MD	UD	BTF	IMPL
Opportunity	Identified	· Solution Needed · Value Established	Viable	-	Addressed	Benefits Accrued
Stakeholders	Recognised	· Represented · Involved · In Agreement	-	-	· Satisfied for Deployment	-
Requirements	Conceived	· Bounded · Coherent	Acceptable	-	Addressed	Fulfilled
Software System		Architecture Selected	-	-	· Demonstrable · Useable · Ready	Operational
Team	Seeded	· Formed · Collaborating · Performing	· Formed · Collaborating · Performing	-	-	Adjourned
Work	Initiated	· Prepared · Started · Under Control	· Prepared · Started · Under Control	-	-	Closed
Way of Working	Principles Established	Foundation Established	-	-	-	-

Table 8: Essence alpha states mapped to full lifecycle

Table 8 maps all the Essence alpha states, and shows that the Essence alphas cover the core phases of the Bank's lifecycle. A detailed analysis of this table is provided in Appendix A2: Analysis of Essence Alpha States.

Mapping the Essence Activity Spaces to the Bank's Lifecycle

Each activity space defined in the Essence kernel takes a set of alphas as input and progresses these alpha's states. The Essence alphas were mapped to the corresponding phase, by comparing the activity space and its completion criteria that progress the alpha to the Bank's defined activities in that phase. The activity spaces that matched were captured in the matrix in Table 9. The matrix

describes how the *Opportunity* alpha is progressed to *addressed* state and the *Stakeholders* alpha is progressed to the state s*atisfied for deployment* through the completion of the activity space *Ensure Stakeholder Satisfaction*. The corresponding activities in the Banks lifecycle occur in the *build*, *test*, and *fix* phase (BTF).

During *high level design* the activity space *Prepare to do the Work* is executed and the *Team* alpha is progressed to *seeded*, the *Work* alpha is *initiated* and *prepared* and the *Way of Working* alpha is progressed to *agreed*. The matrix of Table 9: Activity spaces per phase per alpha, indicates that activity spaces cross phase boundaries in the Bank's lifecycle as activities in each phase add details to the work products that drive the states of the alphas.

Essence Alpha	Phase					
	CE	HLD	DD		BTF	IMPL
		SO	MD	UD	BTF	IMPL
Opportunity	· None	· Explore Possibilities · Understand Stakeholder Needs · Understand Requirements	· Understand Stakeholder Needs · Understand Requirements	· Understand Stakeholder Needs · Understand Requirements	· Ensure Stakeholder Satisfaction (Addressed)	· None
Stakeholders	· Understand Stakeholder Needs	· Explore Possibilities · Understand Stakeholder Needs	· None	· Ensure Stakeholder Satisfaction	· Ensure Stakeholder Satisfaction · Use the System	· None
Requirements	· None	· Shape the System · Understand Requirements · Explore Posibilities · Understand Stakeholder Needs	· Shape the system · Test the System · Understand Stakeholder Needs · Understand Requirements	· Shape the System · Understand Stakeholder Needs · Understand Requirements	· Test the System (Sufficient & Fulfilled)	· None
Software System	· None	· Shape the System · Understand Requirements · Explore Posibilities · Understand Stakeholder Needs · Way of Working · Test the System · Prepare to do the Work	· Shape the System · Test the System	· Implement the System · Prepare to do the Work · Shape the System · Test the System · Understand Stakeholder Needs	· Implement the System · Test the System · Deploy the System	· Deploy the System · Test the System
Team	· Prepare to do the Work	· Prepare to do the Work	· None	· None	· None	· None
Work	· Prepare to do the Work	· Track Progress · Prepare to do the Work · Support the Team	· Plan the Work · Prepare to do the Work · Shape the System · Coordinate Activity · Support the Team	· Track Progress · Shape the System · Prepare to do the Work · Support the Team · Coordinate Activity	· Prepare to do the Work · Support the Team · Track Progress · Shape the System · Coordinate Activity · Test the System	· Coordinate Activity · Track Progress · Prepare to do the Work · Deploy the System
Way of Working	· None	· Pepare to do the work · Support the Team	· Pepare to do the Work	· None	· Support the Team	· Support the Team

Table 9: Activity spaces per phase per alpha

Identification of Practices using the Essence definition of a Practice

The alphas, work breakdown structures, activities, tasks and required work products structured into a process with governance gates, baselines and states, guide the engineers and project managers on what work needs to be completed. These elements do not describe how the work is to be done. The Essence defines the concept of practices to cater for guidance on how to perform a task. These practices describe the techniques that a team or individual will use to complete the work required to progress the state of an alpha.

The Bank's lifecycle activities, work products and technique papers for each identified alpha were reviewed to identify practices that could be mapped to the Essence definition of a practice. This analysis revealed a number of practices:

- *Governance* practice (*Enterprise Architecture* and *Technical*)
- *Concept Evaluation* practice
- *Business Modelling* practice (including process development)
- *Requirements* practice
- *Architecture* practice (*Component* and *Operational*)
- *Project Management* practice, which contains:
    - *Organisational* practices (including milestones, gates and funding)
    - *Social* practices
- *Testing* practice
- *Development* practice
- *Deployment* practice
- *Service Oriented Modelling Architecture* practice(SOMA)

Mapping the alpha, work products, activities, states, roles and checklists resulted in a table that captured the essential parts of the practices. Not all the practices were mapped and a sample of the mapping is illustrated in Table 10, refer to Appendix B: Bank's Alpha Table for a complete mapping:

Bank Alpha	States	Level of Detail	Activities	Work Products	Roles	Checklists
Component (Functional Component)	· Responsibilities Identified · Responsibilities Allocated · Component Specified · Component Developed · Component Deployed	· Logical Application · Specified · Physical	· Outline Conceptual Technical Solution · Outline Technical Solution · Develop Technical Architecture · Develop Physical Architecture · Refine Technical Architecture · Develop Detail Component Design · Develop Solution Software Components · Update Technical Design · Deploy to ETE, QA · Deploy Production Environment · Cutover to Production	· Architecture Overview · Component Model · Conceptual Solution Architecture · Component Design · Component Specification · Source Files	· Application Designer · Application Developer · SOA Designer · System Analyst · Technical Solution Architect	· Architecture Overview · Business Function/System Matrix · Component Design · Component Model · Component Specification · Interface Specification Validation and Verification · Physical Packaging Validation & Verification · Reference Architecture Fit Gap Analysis · Standards Validation & Verification · Subsystem Analysis · UI Design Specifications

Table 10: Component modelling practice

The checklists listed in the matrix do not match the checklists as defined in the Essence. The Essence checklists are defined as the criteria to assess the achievement of the state of an alpha. The checklists defined here are designed to quality check the completed work products, which indirectly will drive the progress of an alpha and its state.

Chapter 5 Analysis of the Research

This chapter provides an analysis of the research work that was concluded to critically analyse the Essence model using the Bank's software development lifecycle. The analysis of the models was approached from two viewpoints to provide a mechanism to assess the Essence model. The first viewpoint assumed the Essence standard is correct and the Bank's lifecycle was assessed against the Essence. The second viewpoint assumed the Bank's lifecycle is the standard and the Essence was assessed against the Bank's lifecycle. The rest of the section is structured as follows:

- Analysis of the essential concepts
- Analysis of the practices
- Analysis of the activity spaces and activities

Analysis of the Essential Concepts

The Bank's core lifecycle is structured and published as a set of tasks and activities that progress the completion of a set of work products through a series of phases. Each activity description has an implied state of completion for the work product that it is acting on. The state of an endeavour executing the lifecycle is assessed through a series of baselines which determine the completion of each engineering phase. The lifecycle does not explicitly define the criteria that can be used to determine the attainment of a baseline. It does, however, specify the work products that are required for a baseline. The lifecycle uses the concept of a slot as a placeholder for these work products. A baseline is therefore composed of one or more slots, which contain one or more work products.

The lifecycle creates each baseline through a series of technical reviews, enterprise architecture peer reviews and financial reviews. The technical reviews have specific criteria that determine the state of the work products and thus the state of the slot.

From this analysis, a set of concepts that are considered important to the Bank can be derived. This leads to the idea that slots could be good candidates for alphas:

- They are important enough to be reviewed in formal reviews
- They are composed of work products that are progressed through the lifecycle
- They are consistent throughout the lifecycle
- They have a set of assessment criteria applied to them in a technical review

Analysing the slots using the concept of an alpha as defined by the Essence model as a framework provides a number of observations into the Bank's governance model. These observations and the corresponding analysis are detailed in Appendix A1: Analysis of Essential Concepts

The initial idea of assessing a slot as a possible alpha was useful in identifying gaps in the governance mechanism and in the set of tasks that progress these work products. It also provided a useful starting point in identifying the concepts considered important in the Bank's lifecycle.

However, a slot, as defined in the Bank's lifecycle, is a placeholder for a set of work products and does not have a set of explicit states. The slot definition allows it to contain work products that are

linked to multiple alphas and additionally it does not map to any activities that will progress its state. For these reasons a slot is not a good candidate for an alpha as defined by the Essence.

The Essence provides a consistent and well represented set of criteria that can be distributed appropriately across the entire Bank's lifecycle. This set of criteria is an effective tool in assessing the coverage and validity of the measures defined in the Bank's governance gates. The use of the criteria in the research highlighted that a number of key concepts are not reviewed optimally in the Bank's lifecycle. Furthermore, the mapping revealed that a number of criteria defined in the Bank's lifecycle are not measuring any input into the gates and thus helped identify missing work products.

The current governance criteria provide a good starting point to develop states for the Bank's core concepts which can be linked to the Essence alphas and thus can be used to progress the Essence alpha states.

The Essence alphas do not map exactly to the baseline model of the Bank's lifecycle, but by mapping the alpha states to the equivalent baseline, listed in Table 8: Essence alpha states mapped to full lifecycle, it becomes clear that the Banks's process has a high focus in driving the completion of a significant amount of work in the *high level design* phase. The high number of states progressed could indicate that too much work is completed in these earlier phases.

The Bank's governance gate assessment provided insight into key states that the *Software System* alpha progresses through, but is lacking in the Essence specification. A key alpha in any software engineering project is the software and the systems on which the software is deployed. The Essence model does not provide states for the design and integration of the software system and its components. A key gap is the link between the development of each component and their assembly into an integrated system. For a large bank with significant integration requirements across a number of disparate systems, the lack of design and integration states would make it difficult to assess the viability of a solution and to synchronise teams. The *Software System* could potentially have *integrated* as a state.

The Bank's lifecycle does not consider the *Way of Working* alpha in any activities, with the exception of the method adoption workshops. It does assess the *Way of Working* alpha indirectly in the governance reviews by reviewing the work products submitted for review against the set of work products defined by the team in the method adoption workshop.

The *Way of Working* alpha state is dependent on what level of the endeavour is being referred to. If referring to the individual component (application) teams, then they would typically have developed a 'way of working' and it would be in use already across all the projects the team is working on. If you are referring to the project team, then it is required to follow a prescribed method, which from an organisation perspective is *in use*. The specific parameters that the project team agree to are discussed and agreed to in the method adoption workshop to develop a specific project method that is tailored from the *in use* method for the needs of the team and type of project.

The method metamodel described previously in *The Bank's Model* section provides a set of core elements that can be considered Bank alphas in much the same way as the slots have been considered. These core elements were reorganised into logical groupings of activities, work products and checklists according to the Essence definition of a practice. This reorganisation was achieved by

reviewing the work products linked to these elements to elicit core concepts, identifying these concepts in the activity descriptions and then reading any technique papers that referred to these concepts. This process led to the identification of a set of concepts that form the foundation of the Bank's lifecycle method. These concepts are:

- Component
- Node
- Data
- Test
- Deployment Unit
- Requirement
- Security
- Business Context
- Business Requirement

These core concepts are the essential things that practitioners in the Bank work with and can therefore be considered as the kernel of alphas for the Bank. Figure 15: Slots modelled as kernel elements, illustrates how these concepts fit together to progress an Essence alpha and how the governance model and the slots could be linked to the Essence alphas. The slot is defined here as a sub-alpha of the Essence alpha and is a placeholder for a set of work products at a level of detail to be assessed at a governance gate. Each Essence alpha can be part of multiple slots and will provide different sets of work products at different levels of detail to each slot.

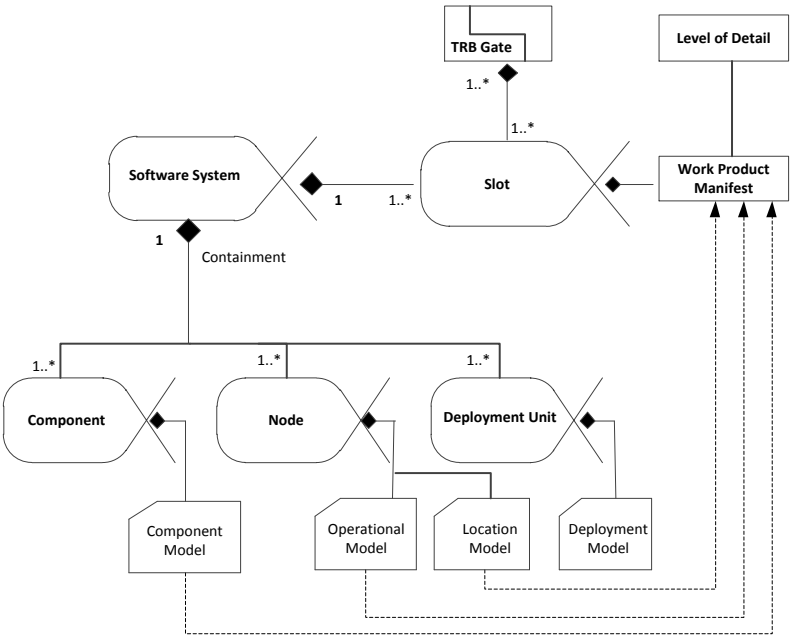


Figure 15: Slots modelled as kernel elements

The sub-alphas, in Figure 15 defined as *Component*, *Node* and *Deployment Unit*, drive the progression of the Essence alpha, *Software System*, via their work products, which will be formally assessed and baselined at each technical review (TRB) gate.

Analysis of Practices

The Bank's lifecycle and method describe a number of useful concepts within a framework that are designed to help improve software quality and the delivery of software solutions into the operational environment. This framework is not explicit in the work breakdown structure, phases and task descriptions. To understand how the activities and work products fit together to produce a quality software system, a practitioner must read a number of technique papers and activity descriptions. To effectively use the lifecycle, a practitioner requires a deep knowledge of the entire model.

Building a method that links all the important concepts in a consistent and easily followed model is not trivial. The SPEM 2.0 model provides a mechanism to easily and consistently document a work breakdown structure. How to capture the core concepts and describing how they link together logically in a work breakdown structure is not clear and not adequately addressed. The Essence provides a mechanism to capture a practice or technique in a way that supports the definition of a set of activities and linked work products to explicitly progress the core concepts of the practice. However it does not provide enough guidance on how to identify alphas and their states and to classify them appropriately. This will make it difficult for software development teams to document their practices.

Analysis of Activity Spaces

Mapping a subset of the activities that progress the Bank's *Component* alpha to the equivalent Essence activity spaces is illustrated in Figure 16: Activity spaces mapped to activities. This mapping highlights some important information:

- It illustrates how the activities driving the development of a software component are linked to each other in a process flow
- It illustrates the completion criteria for an activity and demonstrates how the completion of an activity will drive the development of a software component
- It highlights the gaps in the activities, activity spaces and completion criteria

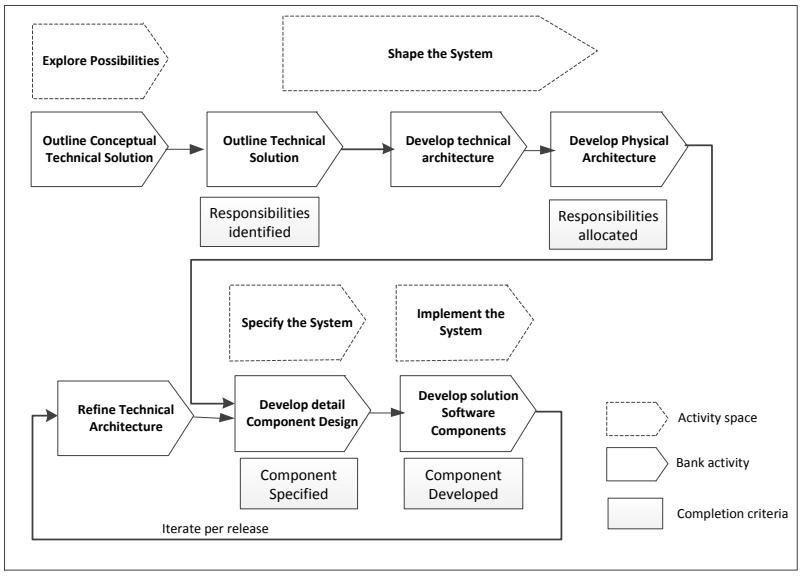


Figure 16: Activity spaces mapped to activities

Analysing the mapping of the Essence activity spaces to the Bank's core concept of *Component* as defined by the Essence specification as a framework provides a number of observations. These observations and corresponding analysis are detailed in Appendix A3: Analysis of Activity Spaces.

A key outcome of this mapping is the difficulty experienced in linking the activities and tasks defined in the Bank's method with the equivalent Essence activity spaces. There are two aspects of this mapping that make this process difficult.

Firstly, there is no easy way to link a newly defined alpha with the existing Essence activity spaces except via the Essence alphas and their completion criteria. The Essence does not provide enough guidance on how to link the states and completion criteria of alphas and sub-alphas to activities and tasks.

Secondly, the Essence activity spaces need to be decomposed into finer grained activity spaces. The alpha states linked to the activity space completion criteria are too high level to enable a useful mapping to the Bank's activities and tasks.

Furthermore, there is no indication of how an activity in the Bank's model progresses a specific work product or core concept. It requires a deep exploration of each activity, each work product that is affected by the activity and any linked guidance material to identify how a work product and its related core concepts are progressed to completion. It is also not clear which completion criteria must be mapped; it can be either the software *Component* state or the work product *level of detail* criteria. A further complication is that the work products assessed do not necessarily map to a single core concept, and thus the progress of these core concepts is not visible.

Chapter 6 Conclusion and Future Work

The objective of this study was to evaluate SEMAT (Software Engineering Methods and Theory) as a framework for software engineering process analysis. To achieve this objective, the SEMAT[9]Essence model was evaluated against an industry lifecycle, specifically an established South African Bank, to gain a deeper understanding of the Essence kernel and identify gaps in the model. This chapter concludes the research described in this research report. The research questions are answered and the contribution of this research report is discussed

Chapter 3 proposes the question and hypotheses that determine the focus of this research.

The research question "In what way can the underlying model of the organisation's software development lifecycle be critically evaluated by mapping the software development lifecycle onto the SEMAT kernel and does the mapping provide insight into the gaps identified?" is answered by discussing its two hypotheses.

1. *The SEMAT framework can be mapped to a custom industry lifecycle and it can be used to describe how the entire process hangs together*
2. *The SEMAT framework is a useful construct in analysing software engineering models and is useful in identifying gaps and guiding the selection of appropriate practices and methods to address these ga*ps

Chapter 4 describes how the industry lifecycle is mapped to the Essence kernel and the Essence concept of a practice can be used to describe the underlying model. The research has shown that the work products, tasks, baselines and governance model that practitioners are expected to work with can be mapped to the Essence model. It is a straightforward exercise to map tasks to alpha activities, and work products to Essence alphas and to document these in the form of an Essence practice. Furthermore, it has been shown that the baselines and the governance *Continuous Assessment Criteria* can be used to align the states of the Essence alphas.

The core elements that drive the underlying metamodel of the Bank's lifecycle can be identified by answering these four questions in the following order:

- What are the essential concepts that practitioners work with (alphas)?
    - Do these essential concepts have states?
    - Do checklists exist to determine the states?
    - What milestones can be expressed in terms of the states of a collection of essential concepts?
    - What collection of work products manifests an essential concept?
- What are the essential activities and tasks that are executed?
    - Which concepts do they drive?
    - Which states do they drive?
    - What criteria are to be fulfilled to measure the completeness of an activity?
- What work products provide evidence of the essential concepts?
    - What activities provide guidance on creating work products
- What patterns are defined? (phases, milestones, gates)
- What practices can be identified?

This set of questions can be used as a simple framework to support organisations in assessing and improving their software development lifecycles.

Chapter 5 describes the findings that resulted from the analysis of the model using the Essence as the framework. The research describes how the use of the framework identified gaps in both the Bank's model and the Essence model.

The Essence alpha *Way of Working* is not optimally supported in the Bank's model, and the corresponding Essence states do not add value to the execution of the lifecycle in a predefined software engineering model.

The *Team* alpha is not managed in the governance model, but is catered for in the method adoption workshops and team support process workshops, which are defined as activities in the lifecycle.

The *Software System* alpha's set of states does not explicitly provide for the design of the components of the software system and does not consider the existing environment that the new or enhanced system will be deployed and integrated into. This reduces the usefulness of the set of states to track the progress of the *Software System* alpha.

Slots are not good candidates for alphas as they are more suited to grouping a set of related work products and do not have the concept of a state. The governance gates apply the assessment criteria to the work products and not to the slots.

Mapping the baselines to the Essence alpha states demonstrated how the alpha states and checklists could be used to standardise the *Continuous Assessment Criteria* and abstract the criteria away from specific work products and tasks.

Identifying potential alphas is straightforward using the framework described above. Relating the newly identified alphas to the Essence kernel alphas is not easy and the Essence specification does not provide any guidance on how to structure the alpha, sub-alpha relationships. Defining an appropriate set of states for a new alpha is not trivial and is easily confused with the level of detail of a work product.

Relating the Bank's tasks and activities to the Essence activity spaces using the completion criteria is very difficult as the completion criteria of the Essence alphas are not linked to a work product completion criterion. There is no indication of how an activity or an activity space progresses the state of a work product.

The work of developing a solid theory of software engineering is only just beginning and more research is required to test the Essence standard. There are a number of areas where future work is suggested to clarify and improve the Essence standard:

- Clear guidelines are required to assist practitioners in using the Essence model to document their practices
- Further work is required in the Essence model to develop a language and metamodel to describe the analysis and design processes used by software engineers
- Further empirical comparisons of other financial services industry lifecycles is required to develop an industry domain model based on the Essence language and concepts

In conclusion this report tests the Essence standard against an operational industry software engineering lifecycle. It demonstrates how the Bank's lifecycle can be described in terms of the Essence kernel. It shows how the Essence alphas and their states provide a useful framework to assess key dimensions that are relevant in an industry lifecycle. The analysis provides a number of insights into these key dimensions and proposes some enhancements to the Essence standard and suggests some improvements that should be considered in the Bank's model.

Appendix A: Detailed Analysis of Observations

A1: Analysis of Essential Concepts

Analysis of Slots and Alphas		
Observation	Evidence	Analysis
There are review gates that have slots as input, but do not have any criterion that assesses the Slot or its constituent work products	The slot *Existing Environment* is documented as a required input into the *Business Requirements Review*, *System Requirements Review*, the Preliminary *Design* Review and the *Critical Design Review*, yet only the *Business Requirements Review* has criteria that evaluate the *Existing Environment* slot work products.	On reviewing the work products associated with the *Existing Environment* slot it is discovered that it only captures the current organisation business context and the existing applications, technology and infrastructure context. Thus this slot does not need to be assessed later on in a project, and there is no valid reason to include it in later reviews. It is recommended to remove the slot in later reviews as it creates unnecessary project overhead to maintain documentation that is not used
There are gates with assessment criteria not linked to a slot	The *Production Readiness Review* contains the criterion *summary available of the baseline requirements versus requirements delivered* and this criterion is not linked to any slot or work product	A set of defined criteria to be used in a formal governance review implies that the criteria must be assessing an attribute that is of some importance to the progression of an endeavour. A criterion that does not have anything to measure may indicate a work product that is missing. On closer inspection there is a critical work product missing. Planning a release would require a clear understanding of the complete set of requirements that must be satisfied, mapped against the set of requirements completed and tested for a particular release, and a plan of when the remaining requirements will be delivered
Mapping the Essence alphas against the slots using the slot work products to guide the mapping reveals that each slot is mapped to at least one Essence alpha	Appendix C: Mapping Slots to Baselines	The slots and their fulfilling work products are considered important concepts and are tracked throughout the Bank's lifecycle. This insight provides support for the claim that the Essence alphas represent the essence of software engineering
Mapping the slots against the Essence alphas reveals that not all alphas map to a slot	The *Team* alpha is not mapped to any slot as it is not addressed by any work products linked to the slots	The *Team* alpha, and to a large degree, the *Work* alpha are not directly addressed by the technical review boards as they are not part of the technical solution being assessed. Enhancing the review board criteria to focus on the *Team* attributes will enable the reviews to highlight potential skills shortfalls and missing domains such as security earlier on in the process, thus avoiding project bottlenecks occurring later in the build and test phases

Analysis of Slots and Alphas		
Observation	Evidence	Analysis
Mapping the slots, assuming they are alphas, to the review gates reveals that not all slots are inputs into a gate	The *Technical Implementation* slot, which covers the actual development outputs and work products is not reviewed by any gate	Initial thinking questioned why the actual development work is not formally reviewed. Further analysis of the process reveals that these outputs are in fact reviewed via the outputs of testing. The test work products are extensively addressed by the *Critical Design Review*, *Testing Readiness Review* and *Production Readiness Review*. This model fits with the Essence alpha states for the alpha *Software System* which is progressed from *architecture selected* directly to *Demonstrable*. The build of the software is considered a mechanism to progress a set of requirements from the state of *conceived* to the state of *fulfilled* and not a state to be achieved on its own merit.
Mapping the slots to the review gates reveals that not all slots are inputs into a gate and challenges our assumptions about what should be measured and the timing of these measurements	The *Requirements Report* slot is not an input into the *Testing Readiness Review* and *Production Readiness Review*.	The requirements are reviewed through the *Test Pack* slot which contains traceability matrices that link test cases back to the use cases. A well-defined set of requirements is of no value to a review that is assessing the readiness of the testing environment and the test cases to begin testing a solution. A work product that maps the set of requirements under review to the test cases that will test these requirements will be useful to assess the appropriateness of the tests and the test coverage
The Bank's governance model does not adequately assess the state of the Essence alphas in later phases	The governance model does not directly measure the state of the *Requirement* and *Software System* alphas in the *implementation* and *deployment* phases. It provides a generic set of criteria that focus on test case approval, completeness and configuration management	The Essence describes a functional test state that assesses the *Requirements* and a quality test state that assesses the *Software System*. Enhancing the Bank's governance model with criteria that focus on both the functional and non-functional tests could improve the test coverage
The gate assessment criteria (CAR) assess and set multiple states for an alpha within a gate	The *Requirements Report* slot's state for use cases is set to *identified uniquely*, *prioritised*, *agreed*, *complete*, *solution independent*, *validatable*, *verifiable*, *testable* and *traceable to scope* in the *System Requirements Review*	The process provides no mechanism to help assess the attainment of each of these states and by assessing them altogether it offers the team no insight into the state of the requirements until much later in the project and places a huge burden on the reviewers to deal with a huge quantity of data for a single review. The Essence provides a set of criteria that determine the state of the complete set of requirements for the endeavour and a separate set of criteria that can determine the state of individual requirements. The validity of assessing the system requirements to this level of detail in a phase that is designed to provide enough information to cost a project needs further investigation
Work products that are used as input into the gates can be composed of other work products. These work products also progress a number of alphas	The *Conceptual Solution Architecture* (CSA), as a work product, will progress its constituent parts to different states. A CSA will have the requirements as *bounded*, the current environment as *validated* and the architecture as *identified*	The CSA is not a good candidate for an alpha as it progresses the states of the *Stakeholders*, *Opportunity*, *Software System* and *Requirements alphas*. The CSA should be defined as a slot or a baseline and not as an individual work product

Analysis of Slots and Alphas		
Observation	Evidence	Analysis
The Essence specification lacks key states and checklists to cater for understanding the existing technical environment.	*Software System* alpha does not cater for understanding the current environment and context into which software will be delivered. It does not address this concept in either its states or checklists	The SDLC requires a current environment to be understood as part of the *Customer Baseline*. The current environment activities and work products are the initial point of departure for driving the *Software System* state. Understanding the current environment is a critical activity for any software endeavour that will either change or interface with an existing operational system. Ignoring the environment could potentially result in significant scope change and requirements analysis in the build and test phases of a project
The Essence specification lacks key development states and checklists that are required for large, multi-system projects	The *Software System* state progresses from *architecture selected* to *demonstrable* and bypasses design, build and integration states	This omission could lead to the state of the alpha as being unknown for a large portion of the lifecycle and the software development and integration progress not being tracked. Tracking the states of multiple systems' design and development progress is a key activity for any large endeavour in a bank. Each system is typically managed by independent teams and not managing these dependencies and their progress could lead to deployment failures and potential rework
The Essence specification provides a simple view of the state of an endeavour from a single perspective. It does not consider multiple teams in complex operational environments	The bank has two levels of teams; the teams that own and manage a technology or application and the project teams that introduce new solutions into the bank. The project teams work with multiple applications and application teams	The way of working for the application team is fundamentally different to the way a project team works. The Bank's lifecycle must cater for both. It must be understood that an application team may simultaneously receive requirements from multiple competing projects and their view of alphas states is not the same as those of any individual project

Table 11: Analysis of slots and alphas

A2: Analysis of Essence Alpha States

Analysis of Essence Alpha States		
Observation	Evidence	Analysis
The Bank's documented model does not cater for states past Implementation	The final state of *Stakeholders satisfied in use* is not evident in the lifecycle	The documented model stops at Implementation and does not have any activities defined to assess this state or progress the alpha to this state. The Bank's model needs to cater for a review of the use of the software after the solution is live and in active use.
Micro design does not progress any Essence alpha	The *Opportunity* alpha is not addressed in *micro design*	This makes sense as no new requirements or opportunities are being defined or explored here. This phase explicitly details each identified component, therefore it does not progress the *Opportunity* state, as the opportunity is now in development. *Micro design* is a development phase as the activities relate to detailed component designs and database designs
Micro design does not progress any Essence alpha	*Requirements* are not progressed in *micro design*	This is correct as the requirements are now being implemented in the design rather than being detailed
Work alpha states are duplicated across *solution outline* and *macro design*	*Work* undergoes a *prepared* state, *started* state and *under control* state for *high level design* and again for *detail design*, BTF and *implementation*	There are two places where *Work* is *prepared*, *started* and *under control*. The first is the end of *concept evaluation*, where an initial scope and business case are tabled and funding for *high level design* is obtained. The second place is at the end of *high level design*, where the final quote for the work is generated and a business case completed for the rest of the project.
Team alpha states are duplicated across *solution outline* and *macro design*	*Team* undergoes a *created*, f*ormed* and *encouraged to collaborate* states for *high level design* and again for *detail design*	There are two, financially driven, phases in which a team is created, formed and encouraged to collaborate. The process requires an initial pre-execution team for *high level design* and a much larger team for *detail design* that will form the execution team for the rest of the project.
Way of Working alpha is only addressed in *solution outline*	*Way of Working* undergoes a *foundation established* state in *solution outline*	The *Way of Working* alpha is addressed in the *high level design* phase through two method adoption workshops. The output of these workshops is a set of activities, work products and an agreed project method that will be used to develop the project plans

Table 12: Analysis of Essence alpha states

A3: Analysis of Activity Spaces

Analysis of Activity Spaces		
Observation	Evidence	Analysis
The Essence activity spaces need to be decomposed into finer grained activity spaces to enable a useful mapping to the Bank's activities and tasks	The Essence activity space *Shape the System* has a completion criterion of *architecture selected* and this criterion in the Essence implies software architecture described at a very detailed technical level.	The Bank requires a number of enterprise architecture and infrastructure architecture decisions to be decided before any software design is completed (refer to Figure 16: Activity spaces mapped to activities). Thus *Shape the System* is composed of three activity spaces in the Bank, where the components are identified, and finally allocated to the appropriate existing system or new system.
The Essence definition of activity spaces in terms of a set of alpha states requires a deep understanding of the concepts underpinning a set of activities or work products to define the completion criteria	Activity space Explore *Possibilities completion* criteria does not have a completion criteria for alternate solutions	The *Outline Conceptual Technical Solution* explores possible solutions to the business opportunity and provides a recommendation on an appropriate solution that balances the enterprise architecture and opportunity requirements with the constraints of the existing operational environment. This identifies a state *alternate solutions identified* gap in the completion criteria for *Explore Possibilities*
There is no indication of how an activity progresses a specific work product in the Bank's lifecycle and thus accurately determine the state of an endeavour	The use case artefact *Detail Key System Use Cases* activity is defined once and reused as is in *solution outline* and *macro design* phases. There is no indication of how the activity progresses the specific work product	Many of the work products and artefacts in the Bank's lifecycle are defined once and reused across activities and many activities are defined once and reused in the process. If the activity had specific work product or alpha states as input and output, then an appropriate work product level of detail and completion state could be defined and a single work product definition and activity definition could be used

Table 13: Analysis of activity spaces

A4: Analysis of Baselines

Analysis of Baselines		
Observation	Evidence	Analysis
The customer area of concern is not supported in design phase	The *Opportunity* alpha is not progressed during the design phase, the essence states are achieved up to *viable* at the *System Baseline*	The next state is met not at the *Test Baseline* as would be expected but at the *Production Baseline*. The *Test Baseline* ensures that all the required tests, test data and infrastructure for testing is defined and available, the test results are assessed at the production readiness review where the *Production Baseline* is reviewed
The *Design Baseline* does not progress the customer and solution area of concern	The *Requirement* alpha is not progressed by the *Design Baseline*.	The *Component Baseline* describes how the components of the system will meet the requirements and will define a system that can be described as acceptable. The *Design Baseline* describes how to build the components that have already described an acceptable system.
The *Software System* alpha is the least supported as defined by the Essence	The *Software System* alpha does not provide any states that deal with *Component Baseline*, *Design Baseline* and *Test Baseline*	*Software System* alpha states move directly from *selecting architecture* which is covered by the *System Baseline* to states that provide evidence of a working system measured by the *Production Baseline*. The *Test Baseline* only deals with test readiness and not with the outcomes of the tests, this is measured as part of the *Production Baseline*
The baselines in the Bank's model do not progress the endeavour aspects of the Essence	*Way of Working* alpha is not addressed at all by the baselines	These alphas are progressed orthogonally to the baselines. In the Bank, the work is specified in a pre-determined schedule and work breakdown structure and each project is expected to follow a prescribed method and the tasks and work products can be tailored to suite the size and complexity of the project. The way of working is predetermined at the organisation level and all teams are expected to follow the process as it is defined thus these states apply more to the implementation of the defined process across all projects
The *Software System Element* sub-alpha covers some of the missing states in the *Software System a*lpha	*Software System Element* sub-alpha sets the state of the alpha to *interfaces identified* at the *Component Baseline*	In the *macro design* phase of the Bank's model, the component model and operational model are specified and the components are allocated detail responsibilities and their key interfaces are agreed and associated to nodes. In the *micro design* phase the individual software components and their agreed interfaces are completely specified.
The baselines in the Bank's model do not address the endeavour aspects of the Essence	*Way of Working* alpha is not addressed by any baseline and work is not adequately tracked.	These alphas are progressed orthogonally to the baselines. In the Bank, the work is specified in a pre-determined schedule and work breakdown structure and each project is expected to follow a prescribed method and the tasks and work products can be tailored to suite the size and complexity of the project. The way of working is predetermined at the organisation level and all teams are expected to follow the process as it is defined thus these states apply more to the implementation of the defined process across all projects

Table 14: Analysis of baselines

Appendix B: Bank's Alpha Table

Bank Alpha	States	Level of Detail	Activities	Work Products	Roles	Checklists
Component (Functional Component)	· Responsibilities Identified · Responsibilities Allocated · Component Specified · Component Developed · Component Deployed	· Logical Application · Specified/Logical · Physical	· Outline Conceptual Technical Solution · Outline Technical Solution · Develop Technical Architecture · Develop Physical Architecture · Refine Technical Architecture · Develop Detail Component Design · Develop Solution Software Components · Update Technical Design · Deploy to ETE, QA · Deploy Production Environment · Cutover to Production	· Architecture Overview · Component Model · Conceptual Solution Architecture · Component Design · Component Specification · Source Files	· Application Designer · Application Developer · SOA Designer · System Analyst · Technical Solution Architect	· Architecture Overview · Business Function/System Matrix · Component Design · Component Model · Component Specification · Interface Specification Validation and Verification · Physical Packaging Validation & Verification · Reference Architecture Fit Gap Analysis · Standards Validation & Verification · Subsystem Analysis · UI Design Specifications
Node (Operational Component)	· Node Identified · Components Allocated · Connections & Interactions Specified · Physical Node Sized · Node Configured	· Logical Application · Logical · Physical · Unsized · Ranged · Sized	· Outline Conceptual · Technical Solution · Outline Technical Solution · Develop Technical Architecture · Develop Physical Architecture · Refine Technical Architecture · Deploy to ETE, QA Environment · Deploy Production Environment · Cutover to Production	· Node Model · Node · Connection · Operational Model · Zone Model · Location Model	· Infrastructure Designer	· Architecture Overview · Candidate Asset List Validation & Verification · Configuration Management · Identify Hardware Software Incompatibilities · Network Conceptual Design Validation and Verification · Network Node Design Validation and Verification · Reference Architecture Fit Gap Analysis · Standards Validation & Verification · Subsystem Analysis · Viability Assessment
Deployment Unit	· Identified · Placed · Connected · Tested	· Logical Application · Logical · Physical	No activities specific for deployment unit, embedded in guidances and templates	· Operational Model · Node Model	· Infrastructure Designer	· None
Data	· Conceived · Resolved · Normalised · Transformed · Implemented	· Conceptual · Logical · Physical	· Develop Data Design · Refine Data Design · Develop Solution Data Components · Deploy ETE, QA Environments · Deploy Production Environment	· Data Migrated · Data Migration Specification · Data Model · Database Transaction Descriptions · Metadata Strategy · Physical Database Design	· Data Modeller	· Data Model · Database Transaction Descriptions Validation & Verification · Physical Database Design
Business Process	· Scoped · Analysed · Gaps Identified · Prioritised · Defined · Tested	· Outlined · Developed	· Capture Existing Environment · Outline Business Solution · Develop Business Solution · Refine Business Solution	· Business Event List · Business Rules · Procedures Document · Process Assessment & Analysis · Process Definition · Process Identification · High Level gap Analysis · Business Function/ System Matrix	· Business Analyst · Process Engineer	· Process Definition Validation & Verification · Process Identification Validation & Verification · Requirements User Report Validation & Verification · Business Direction Validation & Verification · Goal Service Model · Business Function/System Matrix

Table 15: Banks alpha table

Bank Alpha	States	Level of Detail	Activities	Work Products	Roles	Checklists
Test	· Test Strategy agreed · Test Plan Created · Test Environment Ready · Tests Specified · Tests Completed · Tests Successful · Test Cycle: · Scenarios Identified · Test Cases Created · Tests Executed · Test Results Analysed · Test Coverage Satisfactory	· Planned · Identified · Specified	· Outline Test Plan · Develop Test Plan · Conduct Static Testing · Develop Test Specifications · Setup Test Environment · Prepare for Testing · Test Integrated Technical Solution · Perform User Acceptance Testing · Test Production	· Development Integration Test Plan · Master test plan · Operability Test Plan · Solution Verification Test Plan · Static Test Plan · System Test Plan · Test Case · Test Data · Test Environment Configuration · Test Evaluation Summary · Test Execution Plan · Test Findings · Test Log · Test Measurements · Test Plan · Test Script · Test Specification · Test Strategy · User Acceptance Test Plan	· Test Manager · Test Analysts · Tester	· Business Event List Validation & Verification · Acceptance Test Environment · Analyse Test Environment · Determine Level of Tests · Determine Test Focus Area · Early Usability Evaluation & Walkthroughs, Heuristic Reviews · Operability Test Environment · System Test Environment · Systems Integration Test Environment · Test Case · Test Data · Test Environment Configuration Validation & Verification · Test Findings · Test Log · Test Measurements Validation & Verification · Test Script · Test Specification
System requirements	· Identify Requirements · Prioritising Use Cases · Developing High Level Summaries · Developing Detailed Event Flows	· Identified · Detailed	· Outline System Use Case · Describe the System Context · Detail Non Functional Requirements · Elicit and Categorise Requirements · Provide AUMR Checklist · Detail Key System Use Case Scenarios · Document Business Rules · Refine Interface Contsraints ·	· Actor · Assumptions & Dependencies · Authentication & User Management Requirements · Change Cases · Non Functional Requirements · Requirements List · Requirements User Validation Report · Risks & Issues · Security Design Directives · System Context · System Use CaseSystem Case Model · Usability Requirements	Business Analyst	· Change Cases Validation & Verification · Requirements User Report Validation & Verification · System Wide Requirements (FURPS+) · Usability Requirements Validation and Verification · Use Case · Use Case Model · User Groups Validation & Verification · UI Guidelines Validation & Verification · User Profiles Validation & Verification
Work	· Planned · In progress · Completed	None	· Produce Estimation · Draft Execution Project Budget · Document Business Case · Conduct Financial Review Forum (FRF) · Conduct Phase Close Out · Monitor and Control · Develop Solution Development Plans · Detail Solution Development Plans · Update Estimations · Update Lessons Learned	· Configuration Management Plan · Work Product List · WBS · Scope of Work · Release Plan · PM Schedule · Project Estimates · Project Definition · Project Defined Process · Project Budget · Increment Goals · Deployment Plan · Consolidated Sequence of Events	· Project Manager	· Deployment Plan Validation & Verification · Develop High Level Test Schedule · Acceptance Test Plan · Plan validation & Verification · Increment goals Validation & Verification · Master Test Plan · Release Plan Validation & Verification · Static Test Plan · Test Plan · Test Strategy Complete · Test Scope · Usability Design · Evaluation Plan · Validation and Verification · Usability Test Plan · Production Cutover Checklist · Gear Entry Exit Criteria

Table 15: Banks alpha table

Bank Alpha	States	Level of Detail	Activities	Work Products	Roles	Checklists
Business Context	· Identified · Validated · Prioritised	None	· Understand Business Drivers	· Business Direction · Business Drivers · Classified Business Terms · Industry Environment Analysis · Key Performance Indicators	· Business Analyst	· Industry Environment Analysis Validation & Verification · Business Direction Validation & Verification · Business Event list Validation & Verification
Opportunity	· Identified · Validated · Prioritised	None	· Define Problem Statement · Understand Business Requirements · Understand Business Drivers · Define Problem Statement	· Customer Wants and Needs · Problem Statement · Business Case	· Business Analyst	None
Security	none	none	· Consistency Assurance · Conduct Security Design Consistency Assurance · Provide Authentication and User Management Requirements (AUMR) · Prepare Security & Risk Assessment (SRA) · Compile Security Design Directives	· Security Design Directives · SOA LAC Form	· Solution Architect · Business Analyst · Security Designer	· AUMR Checlist · Current Technical Environment · Security Control Checklist · Penetration Testing Preparation Checklist · Security & Risk Assessment
Stakeholders	Identified	None	· Obtain Current Organisation Description	· Stakeholder Analysis · Stakeholder Matrix · User Profiles	· Business Analyst · Process Engineer	None

Table 15: Banks alpha table

Appendix C: Mapping Slots to Baselines

Slot	Work Products	Gates	Essence Alpha	Baselines						
				Customer	System	Logical	Component	Design	Test	Production
Business Requirements	Business Direction, Industry Environment, Business Drivers, Customer Wants & Needs	BRR	Opportunity	X						
Existing Environment	Baseline Survey Report, Current Organisation Description, Process Assessment & Analysis, Process Definition, Technical Environment	BRR,SRR,PDR,CDR	Opportunity	X	X		X	X		
Business Model	Business Event List, Business Rules, Classified Business Terms, Process Definition	BRR,SRR,PDR,CDR	Opportunity	X	X		X	X		
Application Model	Interface Specification,Interfaces, Data Model, Database Transaction Descriptions, Physical Database Designs, Service Design Model	SRR,PDR,CDR	Software System		X		X	X		
Requirements Report	Non-functional Requirements, Actors, System Use Cases, System Use Case Model, Technical Environment, AUMR Checklist, User Profiles, Useability Requirements	SRR,PDR,CDR	Requirements		X		X	X		
Test Approach	Master Test Plan, Test Plan, Test Strategy	SRR/	Work		X					
Architecture Model	Ref Arch Fit Gap Analysis, Security Design Directives, Service Level Characteristics Model, Viability Assessment, Operational Model, Performance Model, Component Model, Architectural Decisions, Architecture Overview, Security & Risk Assessment	SRB,SRR,PDR,CDR, TRR,PRR	Software System		X		X	X	X	X
Standards & Procedures	Standards, Configuration Management Plan, Principles, Policies, Guidelines, Deployment Procedures, Release Candidates, IT Services Strategy, IT Services Detail Design, Deployment Plan, Coding Guidelines, Build Procedures, Application Maintenance Turnover Definition	SRR,PDR,CDR,TRR, PRR	Software System		X		X	X	X	X
Test Pack	Defect Log, Development Integration Test Plan , GEaR Entry and Exit Criteria , Master Test Plan, Operability Test Plan, Solution Verification Test Plan, Static Test Plan, System Test Plan, Test Environment Configuration, Test Execution Plan, Test Findings, Test Log, Test Measurements, Test Plan, Test Specification, User Acceptance Test Plan	SRR,PDR,CDR,TRR, PRR	Software System		X		X	X	X	X
User Experience Model	UI Prototype, UI Guidelines, UI Design Specification, Interactive Concept, Usability Requirements	SRR,PDR,CDR	Requirements		X		X	X		
Technical Implementation	Build, Data Migration Programs, Media Content, Source Files	PRR	Software System							
Project Work	Project Management Schedule, Work Breakdown Structure	Not included in gates	Work							
Project Definition & Scope	Project Definition	SRB,BRR,PRR	Opportunity	X						X

Table 16: Mapping slots to baselines

Appendix D: Mapping CAR's to Slots

Standards & Procedures	Architecture Model	Test Approach	Requirements Report	Application Model	Business Model	Existing Environment	Business Requirements	**Slot**
Standards, Configuration Management Plan, Principles, Policies, Guidelines, Deployment Procedures, Release Candidates, IT Services	Ref Arch Fit Gap Analysis, Security Design Directives, Service Level Characteristics Model, Viability Assessment, Operational Model, Performance Model, Component Model,	Master Test Plan, Test Plan, Test Strategy	Non-Functional Requirements, Actors, System Use Cases, System Use Case Model, Technical Environment, AUMR Checklist, User Profiles, Useability Requirements	Interface Specification Interfaces, Data Model, Database Transaction Desriptions, Physical Database Designs, Service Design Model	Business Event List, Business Rules, Classified Business Terms, Process Definition	Baseline Survey Report, Current Organisation Description, Process Assessment & Analysis, Process Definition, Technical Environment	Business Direction, Industry Environment, Business Drivers, Customer Wants & Needs	**Work Products**
Software System	Software System	Work	Requirements	Software System	Opportunity	Opportunity	Opportunity	**Essence Alpha**
---	1. Solution Options Agreed 2. Approach Ratified 3. Component Model Identified System Architecture Identified Solution Adresses	---	---	---	Business Process: 1. Defined To-Be	Architecture: 1. As-Is Validated Business Process: 1.Identified 2.Validated As-Is 3. Gaps Identified	Business Requirements: 1. Defined 2. Traceable to Scope 3. Agreed/Signed Off 4.Agreed Priority	**BRR States**
Standards: 1. Identified	System Architecture: 1. Agreed(implied) 2. Satisfies System Requirements/Features	Requirements: 1. Testable	Requirements: 1. Identified Uniquely 2. Prioritised 3.Agreed 4. Complete 5. Solution Independent	No CAR	No CAR	No CAR related to this Slot in SRR, but it is an input into the review	---	**SRR States**
No CAR	Architecture: 1. Dynamic Architecture Complete 2. Static Architecture Complete 3. Component Model	---	1. Use cases are Complete 2. Functional Requirements Complete (implied)	1.Database architecture Complete 2. Interaction Diagrams Complete 3. Interfaces Designed (Logical & Physical)	No CAR	No CAR	---	**PDR States**
No CAR	1.Component design Complete 2. Operational Model Design Complete (NFR	---	1. Non Functional Requirements Complete (Implied) 2. NFR Designs Complete 3. Production Data will Support	1.Database Design Complete 2. Interface Design Complete	No CAR	No CAR	---	**CDR States**
1.Problem, Defect ,Change Mangament Procedures Established & Agreed	1.Test Architecture Infrastructure & Software Implemented & Ready	---	---	---	---	---	---	**TRR States**
1.Install Plans Defined 2. Install Procedures Defined	1. Production Infrastructure Agreed (incl Capacity & Scalability)	---	---	---	---	---	---	**PRR States**

Slot	Work Products	Essence Alpha	BRR States	SRR States	PDR States	CDR States	TRR States	PRR States
Test Pack	Defect Log, Development Integration Test Plan , GEaR Entry and Exit Criteria , Master Test Plan, Operability Test Plan, Solution Verification Test Plan, Static Test Plan, System Test Plan, Test Environment Configuration, Test Execution Plan, Test Findings,	Software System	---	Requirements: 1.Acceptance Criteria Identified 2.Validatable 3. Testable	No CAR	Test Plans: 1. Traceable to Component Requirements 2.Traceable to Acceptance Criteria	1.EVT Test Completed 2.Test Data Loaded 3.Test Cases Complete 4.Test Cases Reviewed	1. All Tests Completed Successfully 2. All Tests Signed Off
User Experience Model	UI Prototype, UI Guidelines, UI Design Specification, Interactive Concept, Usability Requirements	Requirements	---	System Architecture: 1. Agreed 2. Satisfies System Requirements/Features 3. UI Standards Agreed	1.Physical Interface Designed(UI)	1. Interface Designs Complete 2. NFR Designs Complete (UI)	---	---
Technical Implementation	Build, Data Migration Programs, Media Content, Source Files	Software System	Not included in gates	---	---	---	---	---
Project Work	Project Management Schedule, Work Breakdown Structure	Work	Not included in gates	---	---	---	---	---
Project Definition & Scope	Project Definition	Opportunity	Scope & Objectives: 1. Defined 2. Verified 3. Agreed 4.Signed Off	---	---	---	---	MTP Deliverables Defined and Agreed Upon

Table 17: Mapping CAR's to slots

References

[1] L. McLeod, S.G. MacDonell, and B. Doolin, “Qualitative research on software development: a longitudinal case study methodology,” *Empirical Software Engineering*, vol. 16, 2011, pp. 430–459.

[2] A.M. French, “Web Development Life Cycle: A New Methodology for Developing Web Applications,” *Journal of Internet Banking and Commerce*, vol. 16, 2011, pp. 2011–8.

[3] O. Cawley, X. Wang, and I. Richardson, “Regulated Software Development–An Onerous Transformation,” *Foundations of Health Information Engineering and Systems: Second International Symposium, FHIES 2012,August, 2012. Revised Selected Papers*, Springer, , pp. 72–86.

[4] H. Van Baelen, “Agile (Unified Process),” *Agile Record*, vol. 6, 2011, p. 22.

[5] J. Appelo, *Management 3.0: Leading Agile Developers, Developing Agile Leaders*, Addison-Wesley Professional, 2010.

[6] P. Clarke and R.V. O’Connor, “The situational factors that affect the software development process: Towards a comprehensive reference framework,” *Information and Software Technology*, vol. 54, 2012, pp. 433–447.

[7] S. Alter, “Is Work System Theory a Practical Theory of Practice?,” *Systems, Signs & Actions*, vol. 7, 2013, pp. 22–48.

[8] B. Boehm and R. Turner, “Observations on balancing discipline and agility,” *Proceedings of the Agile Development Conference*, IEEE, 2003, pp. 32–39.

[9] OMG, “OMG Standard, Kernel and Language for Software Engineering Methods(Essence), Object Management Group (OMG), Document formal/2014-11-02, November 2014,” *http://www.omg.org/spec/Essence/1.0/*.

[10] P. Ralph, “Toxic concepts in systems analysis and design: the systems development lifecycle,” *Proceedings of the 9th AIS SIGSAND Symposium, St. John’s Newfoundland, Canada*, 2010.

[11] B. Boehm and R. Turner, “Management challenges to implementing agile processes in traditional development organizations,” *Software, IEEE*, vol. 22, 2005, pp. 30–39.

[12] J. Vähäniitty and K.T. Rautiainen, “Towards a conceptual framework and tool support for linking long-term product and business planning with agile software development,” *Proceedings of the 1st international workshop on Software development governance*, ACM, 2008, pp. 25–28.

[13] R.G. Fichman and C.F. Kemerer, “Object-oriented and conventional analysis and design methodologies,” *Computer*, vol. 25, Oct. 1992, pp. 22–39.

[14] P. Senge, *The fifth discipline: the art and practice of the learning organisation*, Doubleday, 1990.

[15] I. Jacobson, P.-W. Ng, P. McMahon, I. Spence, and S. Lidman, “The essence of software engineering: the SEMAT kernel,” *ACM Queue*, vol. 10, 2012, p. 40.

[16] “The Authoritative Dictionary of IEEE Standards Terms, Seventh Edition,” *IEEE Std 100-2000*, 2000.

[17] I. Sommerville, *Software Engineering, 9th Edition*, Pearson Addison-Wesley, 2011, .

[18] P. Jalote, *A Concise Introduction to Software Engineering*, Springer, 2008.

[19] H.N. Tran, B. Coulette, D.T. Tran, and M.H. Vu, “Automatic reuse of process patterns in process modeling,” *Proceedings of the 2011 ACM Symposium on Applied Computing*, ACM, 2011, pp. 1431–1438.

[20] M. Kajko-Mattsson, “Maturity is also about the Capability to Conform the Process to the Right Context!,” *Proceedings of the FSE/SDP workshop on Future of software engineering research*, ACM, 2010, pp. 181–186.

[21] M. Morisio, C.B. Seaman, A.T. Parra, V.R. Basili, S.E. Kraft, and S.E. Condon, “Investigating and improving a COTS-based software development,” *Proceedings of the 22nd international conference on software engineering*, ACM, 2000, pp. 32–41.

[22] M. Bloch, S. Blumberg, and J. Laartz, “Delivering large-scale IT projects on time, on budget, and on value,” *McKinsey Quarterly,October*, 2012, pp. 2–7.

[23] T. Dybå and T. Dingsøyr, “Empirical studies of agile software development: A systematic review,” *Information and software technology*, vol. 50, 2008, pp. 833–859.
[24] Applications Leadership Council, “Embrace the Open, ‘Messy’ SDLC,” *Online, https://aec.executiveboard.com/Members/ResearchAndTools/, accessed January 2015*.
[25] A. Myburgh, “Situational software engineering Complex Adaptive responses of software development teams,” *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems (FedCSIS)*, 2014, pp. 841–850.
[26] L. Osterweil, “Software processes are software too,” *Proceedings of the 9th international conference on software engineering*, IEEE Computer Society Press, 1987, pp. 2–13.
[27] P. Kruchten, “A Plea for Lean Software Process Models,” *Proceedings of the 2011 International Conference on Software and Systems Process*, ACM, 2011, pp. 235–236.
[28] J. Coplien, D. Hoffman, and D. Weiss, “Commonality and Variability in Software Engineering,” *From the Trenches IEEE Software*, 1998, pp. 37–45.
[29] OMG, “Software & Systems Process Engineering Meta-Model Specification, Version 2.0, Object Management Group (OMG), Document formal/2008-04-01, April 2008,” *Online, http://www.omg.org/spec/SPEM/2.0/PDF/, accessed 2014*.
[30] B. Elvesæter, M. Striewe, A. McNeile, and A.-J. Berre, “Towards an Agile Foundation for the Creation and Enactment of Software Engineering Methods: The SEMAT Approach,” *Second Workshop on Process-based approaches for Model-Driven Engineering (PMDE 2012), Joint Proceedings ECMFA*, 2012, pp. 279–290.
[31] E. Nardini, A. Molesini, A. Omicini, and E. Denti, “SPEM on test: the SODA case study,” *Proceedings of the 2008 ACM symposium on Applied computing*, ACM, 2008, pp. 700–706.
[32] M. Kuhrmann, D.M. Fernández, and R. Steenweg, “Systematic software process development: where do we stand today?,” *Proceedings of the 2013 International Conference on Software and System Process*, ACM, 2013, pp. 166–170.
[33] P. Kruchten, “Casting Software Design in the Function-Behavior-Structure Framework,” *IEEE Software*, vol. 22, 2005, pp. 52–58.
[34] P. Kruchten, “The frog and the octopus: a conceptual model of software development,” *Online, ArXiv e-prints, http://arxiv.org/abs/1209.1327, accessed June 2015*.
[35] B. Henderson-Sellers, C. Gonzalez-Perez, and J. Ralyté, “Comparison of method chunks and method fragments for situational method engineering,” *Proceedings of the 19th Australian Conference on Software Engineering*, IEEE, 2008, pp. 479–488.
[36] B. Henderson-Sellers, “Method Engineering: Theory and Practice,” *Information Systems Technology and its Applications, 5th International Conference*, 2006, pp. 13–23.
[37] B. Elvesæter, G. Benguria, and S. Ilieva, “A Comparison of the Essence 1.0 and SPEM 2.0 Specifications for Software Engineering Methods,” *Proceedings of the Third Workshop on Process-Based Approaches for Model-Driven Engineering*, ACM, 2013, pp. 1–10.
[38] S. Seema and others, “Analysis and tabular comparison of popular SDLC models,” *International Journal of Advances in Computing and Information Technology*, vol. 1, Jun. 2012, pp. 277–286.
[39] B. Dwolatzky, “Re-founding software engineering practice-The SEMAT initiative,” *Proceedings of the 4th IEEE Software Engineering Colloquium (SE)*, 2012, pp. 1–3.
[40] I. Jacobson, S. Huang, M. Kajko-Mattsson, P. McMahon, and E. Seymour, “Semat—Three Year Vision,” *Programming and computer software*, vol. 38, 2012, pp. 1–12.
[41] I. Meyer B. Jacobson and R. Soley, “Software Engineering Method and Theory - A Vision Statement,” *Online, http://blog.paluno.uni-due.de/semat.org/wp-content/uploads/2012/03/SEMAT-vision.pdf, accessed 2014*.
[42] OMG Submitters, “Essence – Kernel and Language for Software Engineering Methods , Initial Version 1.0, Object Management Group (OMG), Document ad/2011-02-04 , February 2012,” *http://www.omg.org/cgi-bin/doc?ad/2011-02-04/PDF*.
[43] OMG Submitters, “Revised OMG Proposal Submission, Essence-Kernel and Language for Software Engineering Methods, Object Management Group (OMG), Document ad/2013-02-01, February 2013,” *http://www.omg.org/cgi-bin/doc?ad/13-02-01/PDF*.
[44] P.-W. Ng, “Software Process Improvement and Gaming using Essence: An Industrial Experience,” *Journal of Industrial and Intelligent Information*, vol. 2, 2014, pp. 45–50.

[45] C. Péraire and T. Sedano, “State-based Monitoring and Goal-driven Project Steering: Field Study of the SEMAT Essence Framework,” *36th International Conference on Software Engineering, ICSE ’14, Companion Proceedings, Hyderabad, India, May 31 - June 07,*, 2014, pp. 325–334.
[46] K. Smolander and T. Paivarinta, “Forming theories of practices for software engineering,” *Software Engineering (GTSE), 2013 2nd SEMAT Workshop on a General Theory of*, IEEE, 2013, pp. 27–34.
[47] A. Cockburn, “A Detailed Critique of the SEMAT Initiative,” *Online, http://a.cockburn.us/2985, accessed February 2015*.
[48] R. Cornelissen, “Towards a methodology-growing framework,” Msc Thesis, http://fmt.cs.utwente.nl/education/master/236/, University of Twente, 2013.
[49] P. Haumer, “IBM Rational Method Composer: Part 1: Key concepts,” *Rational Edge, December 2005, Online, http://www.ibm.com/developerworks/rational/library/dec05/haumer/, accessed Aug 2015*.
[50] Nedbank, “Nedbank Innovation Lifecycle,” *Online, http://nedbankmethod.nednet.co.za/, accessed Aug 2014*.
[51] P. Spaas, “SDS R3: System Description Standard: Semantic Specification, 2014,” *Online,https://www.ibm.com/developerworks/community/files/app/file/746b578b-fbe0-4c71-ba07-45b72e35cc64, accessed Aug 2015*.
[52] Nedbank, “SIP Governance - TRB - 2012-06-08,” 2012.
[53] Nedbank, “Enterprise Architecture Governance,” *Online, http://team/sites/gt/ea/Governance, accessed Aug*, 2014.
[54] Nedbank, “EA Governance Communication Update,” *Online, http://team/sites/gt/ea/Architecture Services, accessed Aug*, 2014.
[55] Nedbank, “Technical Review Board Information Cafe,” *Online, http://gt.nedportal.nednet.co.za/sites/DnA/DnA/SA/trb/, accessed Aug*, 2014.
[56] Nedbank, “Technical Review Board Workbook,” *Online, http://gt.nedportal.nednet.co.za/sites/DnA/DnA/SA/trb/, accessed Aug*, 2014.