SEF-SCC: Software Engineering Framework for Service and Cloud Computing

Muthu Ramachandran

School of Computing, Creative Technologies, and Engineering
Leeds Beckett University
Headingley Campus
Leeds LS6 3QS UK
M.Ramachandran@leedsbeckett.ac.uk

Keywords: Software Engineering Framework for Service and Cloud Computing (SEF-SCC), Cloud Software Engineering, Service-Oriented Architecture (SOA), Service Computing, Reference Architecture, Service Reuse, Service Component Based Software Engineering (SCBSE), Software Engineering for Service and Cloud Computing (SE-Cloud), Business Process Driven Service Development Lifecycle (BPD-SDL), Business Process Modelling Notation (BPMN), Service-Oriented Architecture Modelling Language (SoaML), Quality of Service (QoS)

Abstract
Service computing and cloud computing have emerged to address the need for more flexible and cost-efficient computing systems where software delivered as a service. To make this more resilient and reliable, we need to adopt software engineering principles and best practices that exist for the last 50 years of experience. Therefore, this chapter proposes a Software Engineering Framework for Service and Cloud Computing (SEF-SCC) to address the need for a systematic approach to design and develop robust, resilient, and reusable services. This chapter presents SEF-SCC methods, techniques, and a systematic engineering process supporting the development of service-oriented software systems and Software as a Service paradigms. SEF-SCC has been successfully validated for the past ten years based on a large scale case study on British Energy Power & Energy Trading (BEPET).

1 INTRODUCTION

Service oriented software engineering and cloud software engineering have emerged to address software as a set of services which deal with user needs. At the same time, we have seen a large number of downfalls due to software failures. This was in part due to the lack of adopting well designed software engineering practices. Now, there is a need to revise these practices given the emergence of service and cloud computing, which will revolutionize the next generation of software engineering.

Software engineering reinforces the application of engineering principles to software development. The major difference between software engineering and other branches of engineering is that software is intangible. However, if there is any malfunction in a software system, it can have tangible effects. Nowadays with the use of software systems and/or software as a service system (SaaS), for high integrity applications such as airborne, financial cloud to medical IoTs, which lead to a very high possibility of a software failure and cyber-attack causing loss of life and financial instability. These software systems are used to conduct our day to day activities and run businesses. With the possibility of a system overload and hackers, this system is far from being secure and safe. Therefore, it is very important to build software which is reliable and trustworthy. Some of the reasons why systems and software fail are:

- Increasing complexity of software systems are cloud driven and distributed as services
- Failure to use software engineering methods, techniques, and process
- Lack of adopting Building Security In (BSI)
- Lack of adopting secure software engineering practices (SSE)

As new software engineering techniques help us to build larger, more complex systems that demand changes rapidly and on-the-fly. Systems have to be built and delivered more quickly; larger, even more complex systems are required; systems have to have new capabilities that were previously thought to be impossible.

Software can be written without using software engineering methods and techniques. However, experience shows that many software failures and breach of security have happened quite often in recent years. Many companies have drifted into adopting software development practices as their products and services have evolved. They do not use software engineering methods in their everyday work. Consequently, their software is often more expensive and less reliable than it should be.

The following are some of the important reasons why Software Engineering needs to be used in Service and Cloud Computing viz:

- Produce clean code which can be easily understood by everyone on the team. With understandability comes readability, changeability, extensibility and maintainability.
- Technical debt in a project should not be accumulated

- Growing complexity of information systems
- Systems and software complexity in terms of design and code is increasing

This chapter presents our approach to evolving a systematic software engineering paradigm known as SEF-SCC which offers new service and cloud software engineering method, process, reference architecture, and techniques and addresses to some of the key research questions using BPMN modelling and simulation during requirements engineering for service computing:

- How do we predict the proposed business requirements and processes will perform efficiently?
- How do we model and simulate business processes?
- How long does it take to find a composable service?
- What is the optimum allocation of services and resources?
- Does the process work well against a large number of service requests?

This is where BPMN modelling and simulation tool helps us to model the business processes and study their performance effectiveness.

2 SERVICE AND CLOUD COMPUTING PARADIGMS

SOA is a formalized way of integrating applications existing (traditional applications and legacy systems) into an enterprise architecture and hence suitability for connecting IOEs. In simple terms, it can be defined as an architecture based on reusable, well defined services implemented by IT components where the components are loosely coupled. Some of the advantages are- minimizes impact of change, platform independence, technology independence, language independence, flexibility to design new solutions from existing IT solutions regardless of where they reside or how they were created.

Service Providers build services and offer them via an intranet or Internet. They register services with service brokers and publish them in distributed registries. Each service has an interface, known as contract and functionality, which is kept separate from the interface. The Service Consumers search for services based on some criteria and when found, a dynamic binding is performed. In this case, the service provides the consumer with the contract details and an endpoint address. The consumer then invokes the service. Services, implemented as Web Services (WS) are delivered using technologies such as eXtensible Markup Language (XML), Web Services Description Language (WSDL), Simple Object Access Protocol (SOAP) and Universal Description Discovery and Integration (UDDI). Enterprise Service Buses (ESBs) build on MOM (message-oriented middleware) to provide a flexible, scalable, standards-based integration technology for building a loosely coupled, highly-distributed SOA. ESBs contain facilities for reliable messaging, web services, data and message transformation, content-based 'straight through' routing. Along

with web services, ESBs are proving to be the major technical enablers for actual SOA projects. Figure 1 shows the benefits of service and cloud computing. It supports multitude of devices, provides distributed services, seamless data, service intelligence, and platform integration. It also allows us to make business decision and predictions based on the performances.

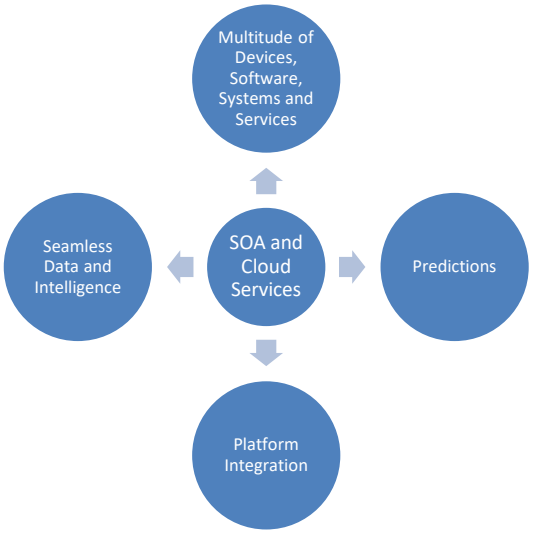

Figure 1 Service and cloud computing benefits

SOA is founded on the notion of Services and each service is characterized by three distinct layers:

- Policies & Service level agreements (SLAs)
- Service Interface and
- Service Implementation, with the first two forming what is generally referred as Service Contract

Every service supports the business/functional capability assigned to it through a set of well- defined operations that are offered on the interface of the service under the specified SLAs (often called QoS Parameters). These operations consist of a set of messages that are exchanged between the consumers and providers during service invocation.

A service is an implementation of a clearly defined business function that operates independent of the state of any other service. It has a well defined set of platform-independent interfaces and operates through a pre-defined contract with the consumer of the service. Services are loosely coupled – a service need not know the

technical details of another service in order to work with it – all interaction takes place through the interfaces.

Data between the consumer and the service are passed in XML format over a variety of protocols. The main protocols that web services use today are SOAP (Simple Object Access Protocol) and REST (Representational State Transfer). While REST uses the existing internet infrastructure (HTTP), SOAP is independent of the network layer and can use a variety of network protocols like HTTP, SMTP and the like.

However, lack of building trust in internet-based technologies remains an outstanding issue for software engineering researchers which has been witnessed the fear of cyber-attacks globally. In addition, the lesson learned from software failures is the lack of application of established key software engineering principles and practices across the lifecycle. Therefore, this chapter presents a novel software engineering paradigm, methods, framework, and tools for service (SOA) and cloud computing. This chapter also distinguishes between software engineering for service and cloud computing and cloud software engineering in the following section. The proposed SEF-SCC method is also applicable to cloud based IoT, fog and edge computing paradigms [1-3].

2.1 SE for Cloud Computing vs Cloud SE

SE for Cloud should focus on engineering approaches to service development process, methods, developing reusable services, systematic approaches to cloud deployment, management, pricing, design for scalability and elasticity that needs to be build-in, tested and deployed by cloud providers.

Cloud SE should focus on engineering approach to developing new services offered by a cloud with emphasis on build-in for scalability, service reusability, and elasticity. Some challenges for Cloud Software Engineering mentioned by Sommerville [4] are:

- Build a software development environment that radically simplifies hosting scientific applications on a range of clouds
- Build applications that make use of the cloud providers Platform as a Service (PaaS) APIs (PaaS) to access common services.
- Challenge in using existing PaaS APIs for computationally-intensive applications, programming models for developing cloud services
- Systems are set up to support web-based applications
- Investigate how to adapt applications that are computation/data intensive to run within the constraints set by the PaaS interfaces from cloud providers
- Application architecture is designed for the cloud: Underlying infrastructure management is done in the program itself, depending on performance and cost requirements, abstractions needed for this etc.

- Need for innovative programming models for the cloud
- Building a PaaS for high performance /throughput computing
- Cloud-aware software development environments

In addition, it is also the key foundation to innovate new abstraction of a service, similar to objects, components, and packages. There are other key challenges such as software engineering approaches to, green cloud computing, in other words, to reduce the amount of power consumed by cloud data centres, emerging IoT and Fog computing applications. It is quite important to make this distinction as the cloud service provider will be able to follow strict engineering principles when they design and install a new cloud service. Our earlier work on designing cloud services using service component models and the design of an independent security service component model can be used as a plug-in [5-6]. This work has also demonstrated the issue of design for software security as a service (SSaaS) [5 & 7]. Therefore, for these above reasons of engineering and achieving required Quality of Service (QoS) (Accuracy, Trustworthy, Safe and Secure), SEF-SCC has been developed and applied to various applications and presented in the following sections.

3 SOFTWARE ENGINEERING FRAMEWORK FOR SERVICE AND CLOUD COMPUTING (SEF-SCC)

SEF-SCC has evolved from various research projects on software components, reuse, and cloud security framework. We believe by adopting a systematic framework for service and cloud computing, the trust in service and cloud computing will increase and be sustainable for all variety of applications from social media, entertainment, medical, financial, and migrating IT systems to the cloud. Figure 2 shows the SEF-SCC framework which consists of methods and design principles supporting service component model, the process, reference architecture, tools, SE-SCC Services, SEF-SCC adoption models, and evaluation & applications.

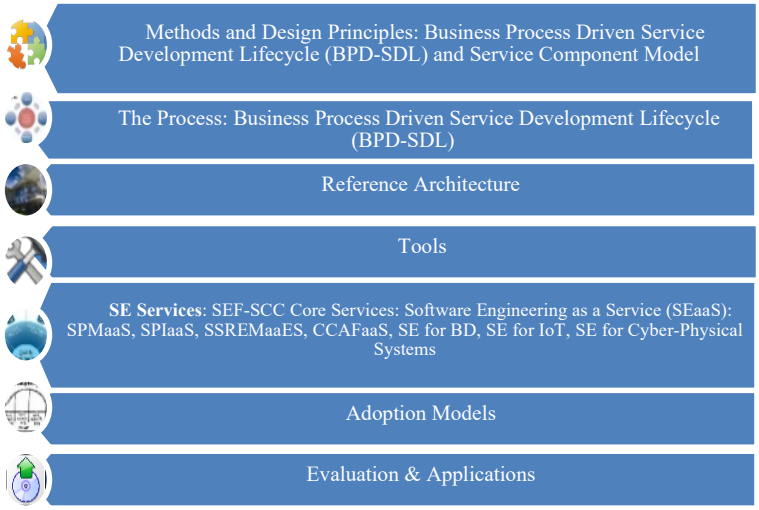

Figure 2 Software Engineering Framework for Service and Cloud Computing (SEF-SCC)

Methods and Design Principles: Service component based method and design strategies using UML components and SoaML have been developed and demonstrated as part of the Business Process Driven Service Development Lifecycle (BPD-SDL). This has been demonstrated with key design principles of separation of concerns, service components, component based architectures, interface design, reuse by service composition for Amazon EC2 case study which involved re-engineering EC2 architecture from the document review process with user guides and published research chapters [5-6] & Software Project Management as a Service (SPMaaS) design with SoaML [9].

Process which provides a full life cycle support starting with service requirements with BPMN modelling and simulation. SEF-SCC provides a Business Process Driven Service Development Lifecycle (BPD-SDL).

Reference Architecture which explicitly supports SOA based design and therefore provides concrete design principles to be sustained for the product line of services.

Tools: a set of tools are proposed such as Visual Paradigm, Bizaghi, and Bonitasoft BPMN modelling and simulation tool for capturing service requirements and to validate business process and performances.

SEF-SCC services: Software Engineering as a Service (SEaaS): Software project management as a Service (SPMaaS) [9], Software Process Improvement as a Service (SPIaaS), Software Security Requirements Engineering Management as a Service (SSREMaaES) [6-7], Cloud Computing Adoption Framework for Financial Cloud (CCAFaaS) [8], SE for BD, SE for IoT, SE for Cyber-Physical Systems.

Adoption models supporting domain-specific application areas based on SEF-SCC has been developed and demonstrated for cloud computing, enterprise security and improvement models. For example, we have developed a Cloud Computing Adoption Framework (CCAF) for cloud security and resiliency framework [10], Enterprise security framework [17], and a framework for internet security [20].

Applications: a set of applications such as Software project management as a Service (SPMaaS) [9], SOA for Big Data Analytics and Business Intelligence [16], SOA for E-Gov [18], have been applied to validate the framework.

Service computing requires multi-disciplinary skills from computing, social science, engineering, and other disciplines to understand what it involves developing a service rather than traditional software which is often not experienced by computer science and software engineering experts. In addition, there are three key challenges in service computing: *service development, service security, and service reuse*. In this context, this chapter proposes three integrated frameworks for service computing as shown in Figure 3. Service development aims to provide the development of a secure, safe, reusable, accurate, correct Quality of Service (QoS) based on engineering lifecycle development stages whereas the service security engineering focuses on Building Security In (BSI) base on the principles of secure service requirements techniques such as application of misuse and abuse service requirements process to the selected service requirement processes using BPMN modelling and simulation to validate consistent processes. The service reuse engineering provides design for reusable services, design with reusable services, application of commonality and variability analysis techniques to selected BPMN services.

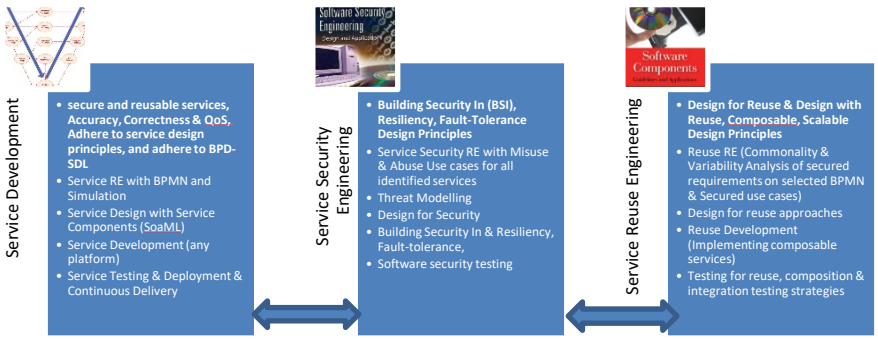

Figure 3 SEF-SCC: Service-Security-Reuse – A Three Integrated Service Engineering Framework

The three-tier integrated framework aims in emphasising the key service design principles of engineering service development, service reuse, and service security by supporting:

- Secure and reusable services,
- Accuracy, Correctness & QoS,
- Adhere to service design principles,
- Adhere to BPD-SDL stages

Each framework can be run in parallel for achieving engineering service development systematically, design for service reuse and design for service security. However, this chapter aims to focus mainly on the service development framework.

3.1 SEF-SCC Process: Business Process Driven Service Development Lifecycle (BPD-SDL)

As part of the SEF-SCC service development lifecycle, we have developed a Business Process Driven Service Development Lifecycle (BPD-SDL) as shown in Figure 4. The BPD-SDL is a key development in addressing the need for a modern software engineering for service and cloud computing (SE-Cloud) and it consists of the following stages which can be iterative and agile, however, BDP-SDL reinforces the engineering approach:

1. Service Requirements – This stage consists of eliciting service requirements from various stakeholders, modelling service requirements using BPMN, conduct simulation, and validate service requirements. Also identify service requirements into functional and non-functional services which forms a set of criteria for performance evaluation of the acquired services. The performance evaluation of time. cost, and resources are the key aspects of service computing and they will be deployed in a cloud which needs to meet those performance criteria. During this stage, it is also a good idea to classify business processes into various business to consumer matrix according to Chen [23]: B2B (Business to Business), B2C (Business to Consumer), C2C (Consumer to consumer), etc. this will be useful to maintain requirements traceability and consistency.

Figure 4 Business Process Driven Service Development Lifecycle (BPD-SDL)

2. Conduct BPMN workflows – This stage consists of categorising services into following classes as shown in Table 1 such as task and entity, and enterprise-oriented services

Service Types / Business Types	Task-Oriented Services	Entity-Oriented Services	Enterprise Services (B2B)
Utility Services			

Business Services			
Co-ordinations and choreographic services			
Human services			

Table 1 Categorising and classifying service requirements workflows

This table is very useful in managing service requirements and to specify detailed WDSL description in the following stage on service interface identification and specification.

3. Service Interface Identification and specification using WSDL – WSDL provides clear template for specifying message-oriented service interface abstractions (InMessage, Operations, OutMessage), Port Type (The bindings) which provides the details of the location and its binding.
4. Service Design with SoaML: SoaML has been developed for service computing by extending the standard UML. SoaML provides different dimensions of the design aspects: Service Contract, Service Component Interface design, and SOA. This is illustrated in the following section on service design.
5. Service cost estimations using service component interfaces (Gupta 2013) and modified COCOMO model for cloud computing (Guha 2013). This stage aims to develop service effort estimation based on sum of the number of interfaces plus adding weighting factors. The second method is a modified form of COCOMO model of cost estimation.
6. Service Implementation - RESTful based services provide a lightweight implementation for service based systems.
7. Service Testing - Service testing should include traditional testing techniques such as unit testing, boundary value analysis testing, integration, and acceptance testing. In addition, for test cloud services as well as SOA, testing should include performance testing, model based testing, Symbolic Execution, Fault Injection Testing, Random Testing, and in particular Privacy Aware Testing as proposed by Priyanka, Chana, and Rana [22]
8. Service Delivery/Deployment – This stage aims to deploy and capture expected QoS and performances as predicted during service requirements simulation using BPMN models.

The BPD-SDL provides a comprehensive set of guidelines based on the principles of service computing and the detailed service elicitation, elaboration, and specification supports consistent design, implementation, testing, and deployment of services.

3.2 SEF-SCC Design: Service Component Based Design Method with SoaML and SOA Based Reference Architecture

SEF-SCC design principles are based on the key concept of separation of concerns as part of the domain modelling process, design for service reusability, and design for service security. To achieve this, SEF-SCC design proposes to use high level abstractions such as virtual machines, service components, and packaging. These design concepts are a natural extension of a software component concept by supporting the notion of provider and requires interfaces. Similarly, the concept of microservices, containers, and serverless programming abstractions, all are based on service components abstraction, as shown in Figure 5.

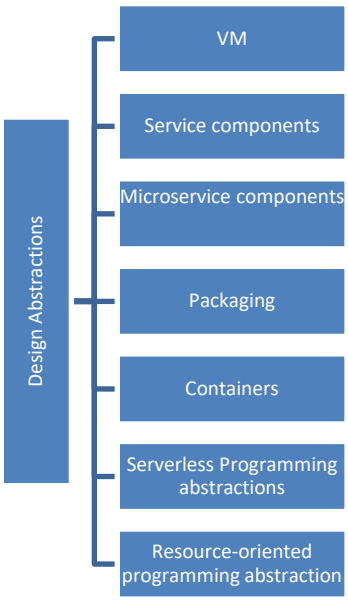

Figure 5 Design Abstraction Principles

SEF-SCC approach to service design emphasises is building on the foundation of design principles such as high level abstraction to achieve loose coupling required for implementing cloud services and SOA based reference architecture. In addition, in order to maximise reuse, scalability, and elasticity, SEF-SCC emphasises on component based design as it provides a natural extension of a software component to implement message-oriented architecture which is required for service and cloud computing. To support, service computing software engineers, SEF-SCC provides a design rationale for selecting appropriate design abstraction

based on established set of design criteria, as shown in Table 2. We believe this is important for making design decisions based on the fundamentals of software engineering to achieve developing a trustworthy cloud service. The design rationale are understanding the definitions of abstractions, loose coupling vs. simpler abstraction, lightweight vs. heavyweight design, implementation, response time, interoperability of multi-clouds and cloud federation, infrastructure support, stateless vs. stateful, end-to-end application vs. flexible packaging, and more efficient cloud resources.

Design Criteria / Design Abstractions	Definitions	High Level Loosley Coupled Abstraction	Lightweight	Response Time	Heavyweight	Multi-cloud Interoperability	Infrastructure	Stateless	Stateful	End-to-end application packaging	Flexible packaging	Simpler cloud abstraction	Faster and more efficient cloud resources	Example platform	References
VM	VM is an abstraction and an instance of a specific OS		√	5-10 Minutes	√		√							Azure, AWS, Google, etc	
Service Components	A lightweight abstraction of a software component and a natural extension of a service supporting interfaces explicitly to connect and compose new services	√	√	Seconds to minutes		√	√	√	√	√	√	√	√	Gravity	Ramachandran (2011)
															Mirandola (2004)
															Gravity
Microservice components		√	√	Seconds to minutes		√	√	√	√	√	√	√	√	Microservices as Docker containers running inside of Kubernetes.	
														Dropwizard	
														Wildfly Swarm	
Packaging		√	√	Seconds to minutes		√	√	√	√	√	√	√	√		
Containers		√	√	In seconds		√	√	√	√	√	√	√	√	Titus: Netflix (AWS) Opensource container management systems such as Kubernetes and Docker Swarm	Loung, Spyker and Bozarth (2018)
Serverless	Serverless provides an abstraction of a service functions without underlying infrastructure in a cloud environment. Specific programming language to be used such as Lambda, Cloud Functions, etc	√	√	In seconds		√	√	√	√	√	√	√	√	Lambda (AWS) Google (Cloud Functions), OpenWhisk (IBM), and Azure	Savage (2018)

Table 2 Design rationale for selecting suitable design abstraction

As discussed earlier, component based design of cloud as well as web services provides a natural dimension to design which incorporates most of the design foundations that are required such as interface based design, high level abstraction of supporting cohesion and loose coupling through separation of concern and flexibility of interface coupling and decoupling to achieve and maximise reuse through service composition. A simple service component model for implementing SEF-SCC as a service (SEFaaS) is shown in Figure 6, as a plugin service component model, and SEF-SCC Service security as a Service (S3aaS) service component model is shown in Figure 7, as a Plugin Service Security Component Model.

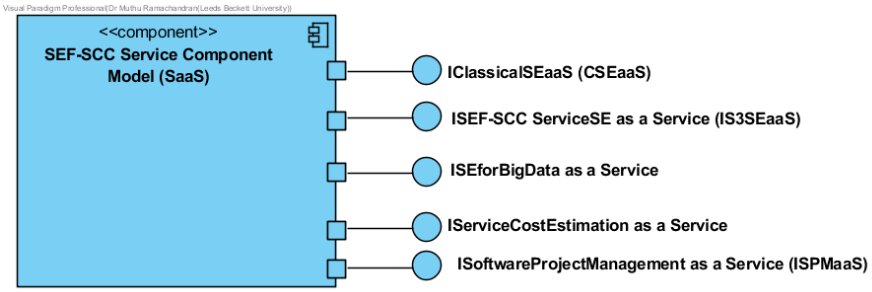

Figure 6 SEF-SCC Service component model (SEF-SCC as a Service (SEFaaS): A Plugin service component model

As shown in Figure 6, a conceptual service design component model, SEFaaS service component model, can be packaged as a container in the cloud environment providing service interfaces (application services) for various SEF-SCC applications discussed earlier such as IClassicalSEaas which aims to provide traditional software engineering as a service on requirements engineering, cost estimation, software project management, design (structured, object-oriented, and software components, and packaging), and testing methods such as unit, integration, and acceptance. Service interface on ISEF-SCC as a service (I3SEaaS) aims to provide SEF-SCC as a service. Service interface on ISEforBigData as a Service which aims to provide software engineering for big data services of streaming, analysing, evaluating, pruning, visualising, analytics, and predictions.

Cost estimation for service computing is relatively a new area which remains unexplored in research. However, Gupta [14] has proposed a service point estimation model which aims to calculate the sum of service component interfaces with some weighting factor. Therefore, the service interface, IServicePointCostEstimation as a Service (ISPCEaaS) aims to provide an autonomic service for computing the service cost and therefore achieving the estimation of service complexity analysis before actual implementation and the performance characteristics can also be evaluated and measured against initial BPMN simulations. In addition, Guha [21] has proposed a modified cloud COCOMO model with weighting for cloud computing projects are: a = 4, b = 1.2, c = 2.5, d = .3. Therefore, the effort and cost estimation equations are:

$$Cloud\ computing\ project\ effort\ applied\ (EA) = a \times (Service\ Points)^{b}\ (Human\ Months) \quad \text{---- (1)}$$

$$Cloud\ computing\ development\ time\ (dt) = c \times (Effort\ Applied)^{d}\ (Months) \quad \text{---- (2)}$$

$$Number\ of\ Service\ Development\ Engineers\ Required = Effort\ Applied\ (EA)/Development\ Time\ (dt) \quad \text{---- (3)}$$

The equations 1-3 provide cloud project effort and cost estimations based on service points which is the sum of all interface function points.

Similarly, the service interface, ISoftwareProjectManagement as a Service (ISPMaaS) aims to provide support for classical project management activities such as requirements management, scheduling, planning, managing resources, and efficient resource management.

In addition, the best practice service component design principles are:

- Use port concept to define all interfaces
- Always follow best practice guidelines on interface design principles with service components which is a natural extension of a service

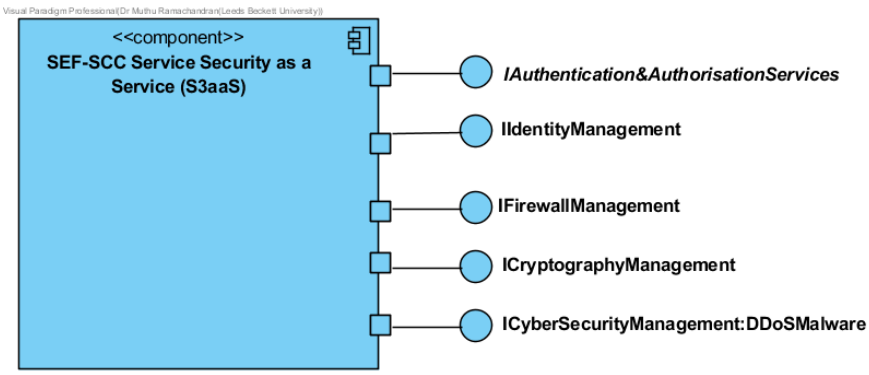

Figure 7 SEF-SCC Service security as a Service (S3aaS): A Plugin Service Security Component Model

As shown in Figure 7, a conceptual service design component model, SEF-SCC Service security as a Service (S3aaS) service component model, can be packaged as a container in the cloud environment provides service interfaces (provider application services) for various SEF-SCC service security applications such as IAutentication&Authrisation, IIdentityManagement, IFirewallManagement, ICryptograhyManagement, and ICyberSecurityManagement for services against malware, adware, and DDoS attacks, etc.

As part of our best practice design guidelines, BPMN modelling and simulation for capturing and validating requirements, SOA design method consists of designing service components, designing SOA based reference architecture and designing SoaML SOA, and finally mapping service components to SOA reference architecture. The following section presents SOA based SEF-SCC architecture design and best practices.

3.3 SEF-SCC SOA Based Reference Architecture

SOA is a design method based on service provider, service consumer, and service directory model which has been established as a proven design method for service computing. One of the key importance of proposing SEF-SCC reference architecture is to standardise the evolution of SOA based software products, and cloud based services for all software as a service paradigm design and implementation. SEF-SCC architecture best practice guidelines are:

In addition, a reference software architecture provides a generic structure for a class of software systems and software product line engineering to achieve a standard practice [24]. Similarly, NIST reference architecture for cloud computing defines to provide an overall framework and to communicate accurately across cloud services and hence providing support for vendor-neutral design [25]. In this context, SEF-SCC has developed a reference architecture based on SOA framework (RAG-SCC) as shown in Figure 8.

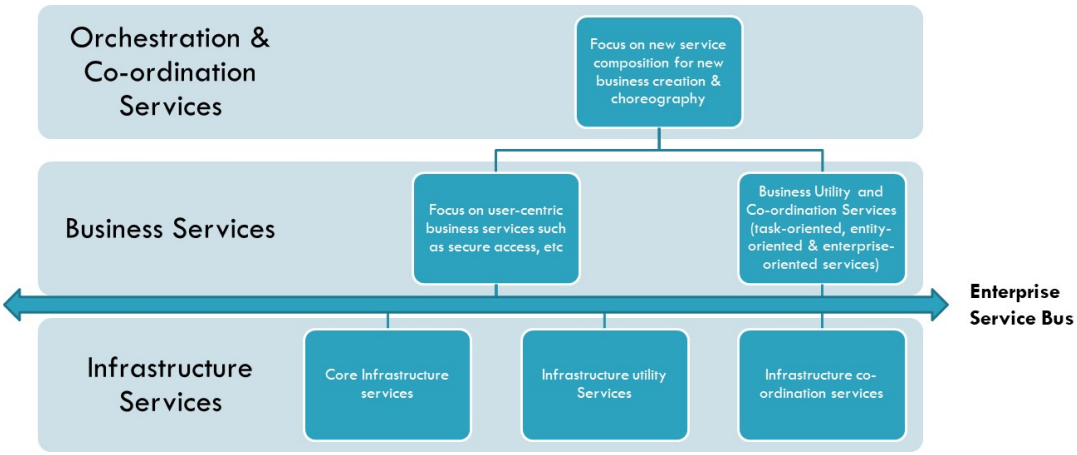

Figure 8 Reference Architecture for Generic Service and Cloud Computing (RAG-SCC)

RAG-SCC follows the design principles of SOA, provides three layers architectural model, the top layer is orchestration and coordination services which focuses on new service composition for new business creation & choreography, the middle layer is known as the business layer which provides container for all business services components and their connections and focus on user-centric business services such as secure access, Business Utility and Co-ordination Services (task-oriented, entity-oriented & enterprise-oriented services), the enterprise service bus which is the back bone of SOA architecture by providing common

communication pathway (supporting the concept of unified modelling) for all services from various layers, and the bottom layer is known as the infrastructure layer which provides Core infrastructure services, Infrastructure utility Services, Infrastructure co-ordination services. We also recommend service security are build-in security components as plugins and in all layers in the reference architecture.

The next step in the SEF-SCC design process involves designing SOA design with SoaML which is one of the best practice for providing a completed design as shown in Figure 9 for all SEF-SCC services such as the consumer are multi-cloud users, the provider is multi-cloud agent with cloud software engineers, there are SLAs for both parties to connect and communicate for using and composing new services, the sample services are:

- Cloud Project Management as a Service (CPLaaS)
- Cloud Reuse patterns as a Service
- Software Security engineering as a Service
- CloudML as a service
- Software Process Improvement as a service
- Cloud RE as a Service
- Cloud Project Cost Estimation as a Service
- Design for Service Reuse as a Service
- Cloud Testing as a Service
- BPMN as a service
- Continuous Delivery as a Service
- Cloud Service Design as a Service

There are plenty of services that can be co-ordinated and composed for new services and therefore it is one of the best practices for service design offered by SEF-SCC design approach.

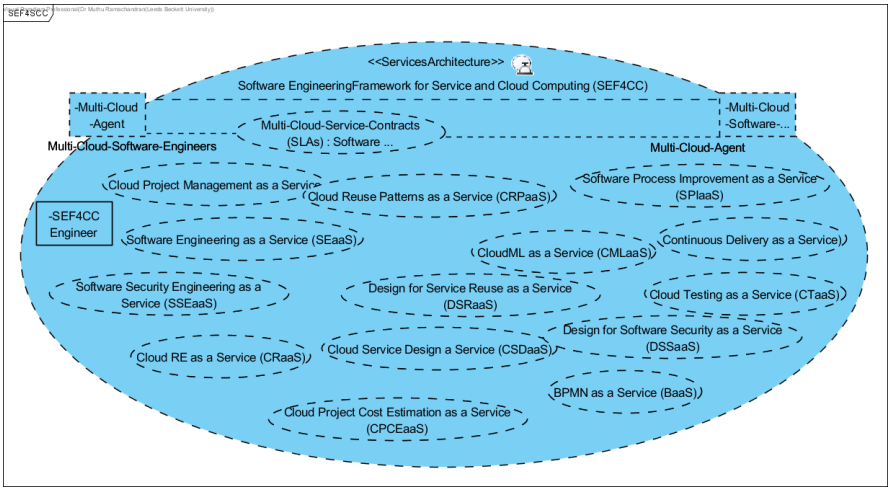

Figure 9 SOA Design for SEF-SCC Services with SoaML

After SOA design with SoaML, the final step in the design process is to compose the classified service as shown in Table 1 and map them into the reference architecture as shown in Figure 10.

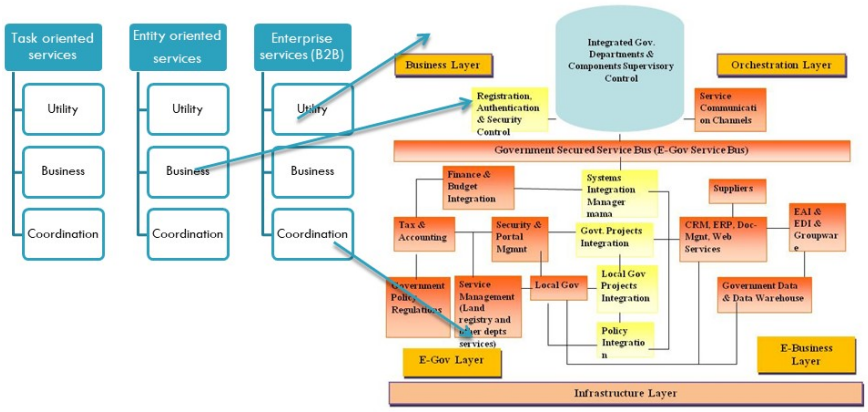

Figure 10 Mapping Services to SEF-SCC Reference Architecture

The mapping of services into the SOA architecture requires architectural design skills to evolve the design rules that can be embedded into the services and the architectural layers. The following section shows a proposed automated CASE tool for developing secure cloud services based on the best practice guidelines presented in this chapter.

3.4 Autonomic CASE Tool for Service Computing

Autonomic service computing is emerging faster as they are SOA based which supports autonomic computing using reuse of knowledge and services. For example, Bellur [26] proposes an autonomic service-oriented middleware for IoT based systems (AUSOM) by applying MAPE-K loop (monitor, analyse, plan, act using stored knowledge). Similarly, Bocciarelli [27] proposed the design of Business process modelling and simulation as a Service (MSaaS). Therefore, with the emergence of AI, we believe, we can develop an autonomic system as shown in Figure 11 based on Model services, Design and develop services, Test and Deploy Services using stored knowledge, best practices of SEF-SCC, and reuse of services by composition.

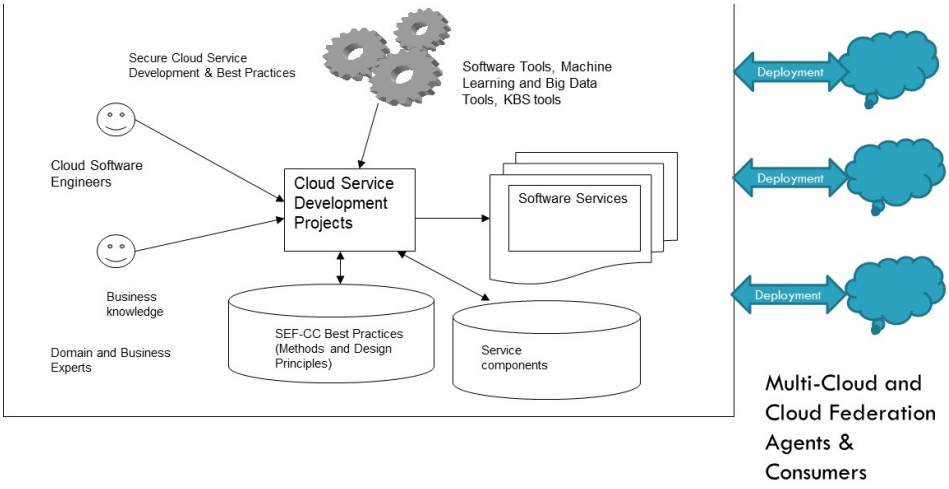

Figure 11 Service CASE Tool for Developing Cloud Services

As shown in Figure 11, SEF-SCC CASE aims to provide support for a complete business driven service development lifecycle discussed in this chapter (BD-SDL) with service requirements modelling and simulation with BPMN engine, knowledge discovery engine provides best practices on service development, and service component reuse.

4 SEF-SCC FRAMEWORK EVALUATION CASES STUDY ON BEPET AND SEF-SCC APPLICATIONS

This section aims to provide how BD-SDL and SEF-SCC best design practices have been evaluated using a case study on BEPT over ten-year period with more

than 12 software engineers who are trained on the method and have been strict to use the BD-SDL principles. The chart 1 and 2 shows the empirical evaluation of the method proposed in this chapters.

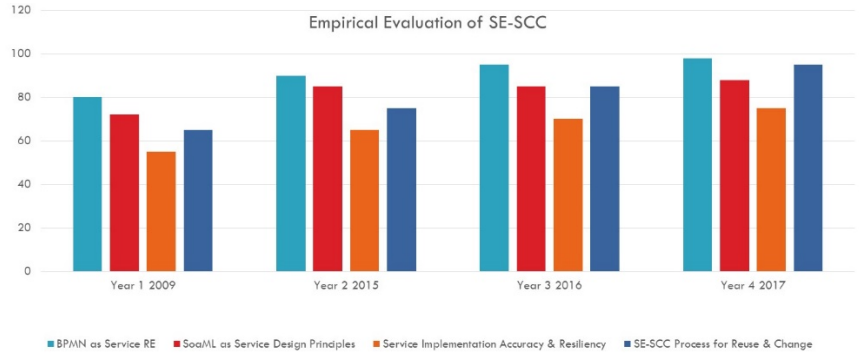

Chart 1 Evaluation Results on the Applicability of SEF-SCC Service Development Lifecycle

Chart 2 shows the number of service components developed for SPMaaS, SSEREaaS, SPIaaS, and SEaaS applications designed as part of the SEF-SCC applications.

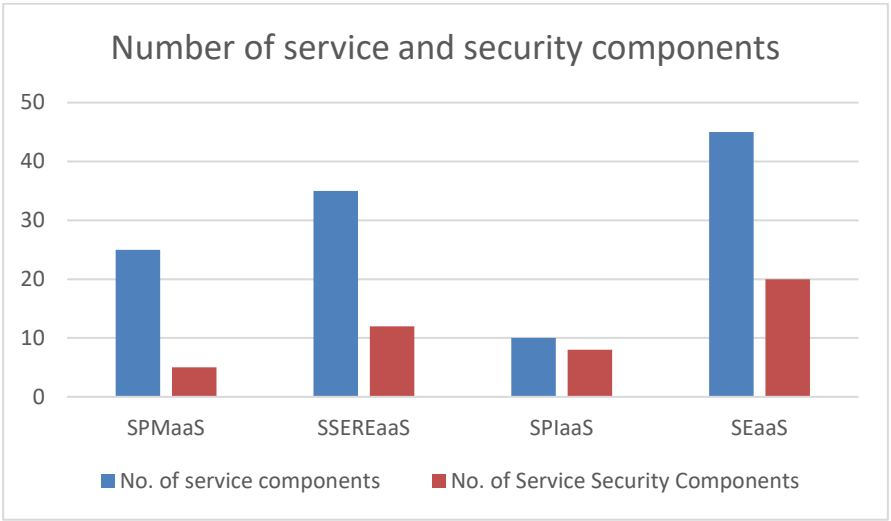

Chart 2 Number of service and service security components for SPMaaS, SSEREaaS, SPIaaS, and SEaaS

There are further research challenges ahead in developing autonomic CASE presented in this chapter as well as developing numerous applications as a service.

Conclusion

Service computing has emerged to offer greater business flexibility and globalising economy as well as social mobility by connecting and communicating with people. This chapter presented our approach to Software Engineering Framework for Service and Cloud Computing (SEF-SCC) to address the need for a systematic approach to design and develop robust, resilient, and reusable services. This chapter presented SEF-SCC methods, techniques, and a systematic engineering process supporting the development of service-oriented software systems and Software as a Service paradigm. As part of the SEF-SCC design process and its best practice design guidelines have been valuable assets for creating an engineered approach to SOA and cloud services and also have been used to train software engineers for more than 10 years for developing their skills in service computing.

References

1. Hu, P., et al. (2017) Survey on fog computing: architecture, key technologies, applications and open issues, Journal of Network and Computer Applications 98 (2017) 27–42.
2. Mahmud, R., Ramamohanarao, K., Buyya, R (2010) Latency-aware Application Module Management for Fog Computing Environments, ACM Transactions on Embedded Computing Systems, Vol. 9, No. 4, Article 39, March 2010
3. Subramanya, T., et al. (2017) A practical architecture for mobile edge computing, IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)
4. Sommerville, I (2012) Challenges for cloud software engineering, http://pire.opensciencedatacloud.org/talks/Cloud-Software-Challenges.pdf
5. Ramachandran, M (2011) Software components for cloud computing architectures and applications, Springer, Mahmood, Z and Hill, R (eds.) Cloud Computing for Enterprise Architectures, www.springer.com/computer/communication+networks/book/978-1-4471-2235-7
6. Ramachandran, M (2012) Software Security Engineering: Design and Applications, Nova Science Publishers, New York, USA, 2011, **ISBN:** 978-1-61470-128-6, https://www.novapublishers.com/catalog/product_info.php?products_id=26331
7. Ramachandran, M (2016) Software Security Requirements Engineering and Management as an Emerging Cloud Service, International Journal of Information Management, 36(4), 2016, Elsevier, Volume 36, Issue 4, August 2016, Pages 580–590, doi:10.1016/j.ijinfomgt.2016.03.008
8. Ramachandran, M and Chang, V (2014) Modelling Financial SaaS as Service Components, Intl workshop on Emerging Software as a Service and Analytics (ESaaSA 2014), The 4th International Conference on Cloud Computing and Services Science, CLOSER 2014, 3-5th April, Barcelona, Spain
9. Ramachandran, M and Chuagle, V (2016) Software Project Management as a Service (SPMaaS): Perspective and Benefits, Software Project Management for Distributed Computing: Life-Cycle Methods for Developing Scalable and Reliable Tools, Mahmood, Z (ed.), Springer, 2016
10. Chang, V and Ramachandran, M (2016) Towards achieving Cloud Data Security with the Cloud Computing Adoption Framework, IEEE Transaction on Service Computing, Issue No.01 - Jan.-Feb. (2016 vol.9), pp: 138-151.

11. Delgado, A., et al. (2011) Business Process Service Oriented Methodology (BPSOM) with Service Generation in SoaML, Advanced Information Systems Engineering - 23rd International Conference, CAiSE 2011, London, UK, June 20-24, 2011. Proceedings
12. Savage, N (2018) Going Serverless, Communications of the ACM, February 2018, Vol. 61, No. 2
13. Leung, A., Spyker, A., and Bozarth, T (2018) Titus: Introducing containers to the Netflix cloud, Communications of the ACM, February 2018, Vol. 61, No. 2
14. Gupta, D (2013) Service Point Estimation Model for SOA Based Projects, http://servicetechmag.com/system/application/views/I78/1113-1.pdf
15. Mahmood, Z and Saeed, D (eds) (2013) Software Engineering Framework for Cloud Computing Paradigm, Springer, 2013
16. Ramachandran, M (2016) Service-Oriented Architecture for Big Data and Business Intelligence Analytics in the Cloud, Computational Intelligence Applications in Business Intelligence and Big Data Analytics" Sugumaran, V. Sangagaiah, A and Thangavelu, A (eds), CRC Press, (Taylor & Francis Group)
17. Ramachandran, M (2014) Enterprise Security Framework for Cloud Data Security, Book chapter "Delivery and Adoption of Cloud Computing Services in Contemporary Organizations, Chang, V (ed.) IGI Global
18. Ramachandran, M., Zaigham, M., and Pethu, R (2014) Service Oriented Architecture for E-Government Applications, Emerging Mobile and Web 2.0 Technologies for Connected E-Government, IGI Global.
19. Ramachandran, M (2013) Business Requirements Engineering for Developing Cloud Computing Services, Springer, Software Engineering Frameworks for Cloud Computing Paradigm, Mahmood, Z and Saeed, S (eds.), http://www.springer.com/computer/communication+networks/book/978-1-4471-5030-5
20. Ramachandran, M and Mahmood, Z (2011) A framework for internet security assessment and improvement process, chapter 13, Knowledge Engineering for Software Development Life Cycles: Support Technologies and Applications (ed. Ramachandran, M), IGI Global Publishers, USA, ISBN-13 978-1609605094.
21. Guha, R (2013) Cloud COCOMO/Modified COCOMO for Cloud Service Cost and Effort Estimation Technique: Impact of Semantic Web and Cloud Computing Platform on Software Engineering, Mahmood, Z and Saeed, D (eds) (2013) Software Engineering Framework for Cloud Computing Paradigm, Springer, 2013
22. Prinyanka, Chana, I., and Rana, I (2012) Empirical evaluation of cloud-based testing techniques: a systematic review, ACM SIGSOFT Software Engineering Notes, May 2012 Volume 37 Number 3
23. Chen, S (2005) Strategic Management of e-Business, Wiley, 2nd Edition
24. Angelov, S Paul Grefen, Danny Greefhorst, (2012) A framework for analysis and design of software reference architectures, Information and Software Technology 54 (2012) 417–431
25. Liu, F (2011) NIST Cloud Computing Reference Architecture, NIST Special Publication 500-292
26. Bellur, U (2017) AUSOM: Autonomic Service-Oriented Middleware for IoT-Based Systems, IEEE 13th World Congress on Services
27. Bocciarelli, P. et al (2017) Business process modeling and simulation: state of the art and MSaaS opportunities, SummerSim '17 Proceedings of the Summer Simulation Multi-Conference, Bellevue, Washington — July 09 - 12, 2017

Park, S., Lee, S., and Park, Y.B (2015) Best Practices in Software Engineering for SaaS-Cloud Era, Park, J.J. et al (eds) Computer Science and its Applications: Ubiquitous Information Technologies, Springer, DOI: 10.1007/978-3-662-45402-2_31