A self-learning solution for effective service modelling and portfolio management

Rasha Daoud
rasha.daoud@gmail.com

September 4, 2019, 67 pages

Research supervisor: dr. Z. (Zhiming) Zhao, z.zhao@uva.nl
Host supervisor: F. ter Beek, felix.terbeek@stachanov.com
Host organisation: Stachanov Solutions and Services, https://www.stachanov.com/

Universiteit van Amsterdam
Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Master Software Engineering
http://www.software-engineering-amsterdam.nl

Abstract

Software companies aim to continuously improve the efficiency of delivering solutions to customers and reduce their software maintenance cost and maintain sustainability as Software evolves. Service Oriented Architecture (SOA) cuts on development costs and efforts by allowing developers to model their software assets with standardized interfaces and to reuse them in new solutions as building blocks. A Service Portfolio documents the service assets and their evolution managed by the company, including services in the pipeline, retired services, and 3rd-party services which are an integral part of the service offerings to customers. It provides a common information source for developers, sales, customer support and other roles in the company.

The functionality and the granularity of the entities in the portfolio have a direct impact on the reusability and sustainability of those assets. When developing various SOA solutions and with the continuous development and delivery in Agile, design decisions must be always taken to slice and realize the required scope of functionality into services and components. Modelling existing assets as services with well-defined granularity can not only significantly leverage them in the design loop, and optimize the development costs, but also improve the maintenance efficiency of the service portfolio by minimizing the number of less reusable inventories and variations.

Existing methods focus on identifying and creating services in the design loop, but don't consider integrating proper granularity and development cost indicators from the portfolio in the modelling decision pipeline and the evolution of the portfolio. We thus proposed a decision support approach that uses a collaborative modelling technique for both legacy/ existing assets and new required business functionalities and uses variation analysis activities in the design loop to discover assets for reuse. Combining this approach with the continuous development and delivery makes it work as a self-learning design support system that continuously aligns available software assets with business needs and helps companies evolve their service portfolio in the right direction for the market.

Contents

Chapter 1

Introduction

In this chapter, we briefly present background information for this thesis. We describe the research motivation, the problem we propose to solve, and how this problem manifests itself at the host organization. Then, we present our research structure and introduce our proposed solution and our contribution to the targeted research area. Finally, we present the outlines of the following chapters.

1.1 Background

1.1.1 Service Oriented Architecture (SOA)

SOA is an architectural design style which enhances the component-driven architecture approach. It allows separate services, namely, software units that are created using different technologies and hosted on different servers to be used to assemble a software solution. The service which forms the basic building block in SOA applications is usually self-contained and encapsulates certain context to perform a task. It is provided to other application components and different applications through certain communication protocols and may consist of other underlying services (See Fig 1.1). This means a service in SOA can be atomic or combines other services by itself. Having services distributed, loose-coupled, and platform-independent in SOA gives means to standardize the service integration, automate the deployment, and enhance reusability and scalability of software assets and both software and business agility. It also helps to extend applications life span and facilitates replacing or upgrading the building blocks [29, 32].

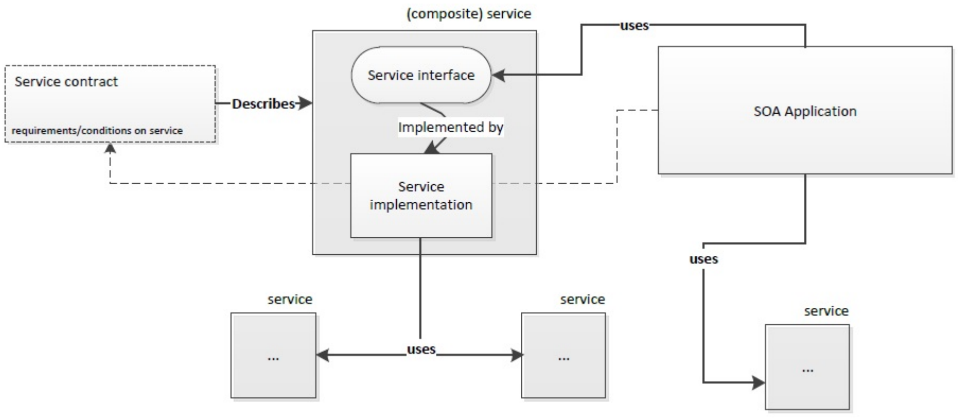


Figure 1.1: Simplified overview: services in SOA application

SOA has many service design principles to unlock its benefits, including flexibility in introducing requirements and technology changes and reuse of existing services to cut development costs. To reduce development costs for new SOA applications, the reusability of existing service assets through service composability is key. The relation between SOA design principles and service reusability as one enabler for SOA benefits is shown in Fig 1.2. Besides that, standardization of the service description, so-called service contract across the whole service catalog is important to support services reuse and ease the discoverability and composability.

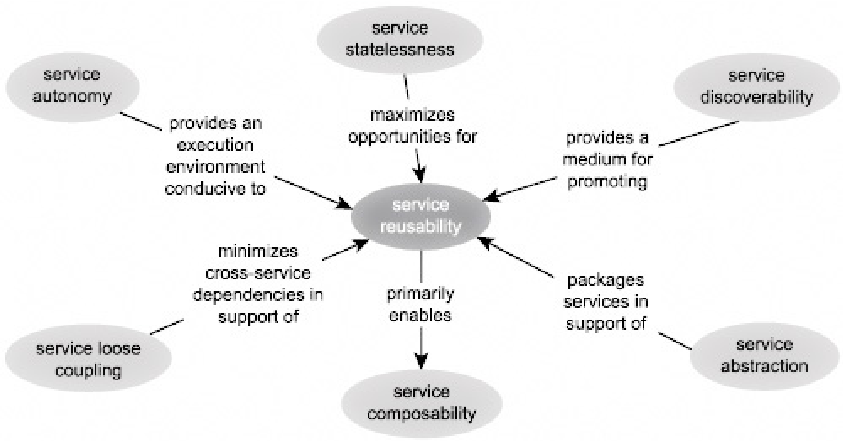

service statelessness
service autonomy
service discoverability
provides an execution environment conducive to
maximizes opportunities for
provides a medium for promoting
service reusability
minimizes cross-service dependencies in support of
primarily enables
packages services in support of
service loose coupling
service composability
service abstraction

Figure 1.2: The relation between SOA principles and service reusability

Service granularity and SOA modelling Service granularity refers to the scope of functionality or the size of the functionalities one service contains. In other words, it is the quantified encapsulated functionality in one service [13]. Choosing granularity of services is a critical task when identifying services in new SOA solutions, as services should have a certain scope of functionality to be loosely-coupled and reused to realize various business functionalities and processes as well as easier to change when requirements or technology change. On the other side, modelling software assets in SOA plays an important role in defining services and choosing proper granularity. To cut on development efforts when slicing new set of functionalities in the design loop, existing assets should have a standardized interface and should be modelled using a suitable modelling technique to enable development teams leveraging assets to interpret them correctly and reusing them. This also helps to reduce future maintenance costs and minimize the number of less reusable services and overlapping services in service inventories, which is crucial for business.

Existing modelling methods propose different types of modelling languages and ontology to model SOA artifacts, including identified services in SOA solutions and their relationships. However, those methods such as SOMA [3] which model SOA solutions don't consider modelling legacy systems as services and modelling available service assets in inventories in a collaborative modelling technique as requirements and assembled solutions to include them in their existing assets analysis and speed up leveraging and interpreting them in design loops. Without having such a modelling method, development teams that deliver solutions iteratively or work on different products may introduce improper granularity decisions that don't consider the evolution of assets and miss the chance of reusing services assets and maintain their SOA solutions to stay reliable and resilient to change and evolve every iteration.

1.1.2 Service lifecycle and portfolio management

SOA Governance: While maintaining a suitable set of services for customers at a desirable cost is key in companies, SOA governance is an extension of IT governance that focuses on managing service lifecycle, usage, and registry [11]. To succeed in SOA, nothing in the software company should exist in isolation. Leveraging existing assets and data by provisioning them as shareable services is important. For instance, IT teams are expected to leverage services to quickly deliver solutions or introduce changes to them. SOA governance empowers people to work in a collaborative manner efficiently by defining their roles and responsibilities clearly and managing different processes that are related to service lifecycle. Otherwise, there is a little chance to benefit from adopting SOA and for sustaining service assets and inventories that align SOA services with business needs and the company's strategies.

Service portfolio and service portfolio management: A service portfolio is a repository that documents three important domains at the software company and offers an overview of their statuses. Those domains are as follows:

- Service catalog: it includes the already exposed and available services and should provide a clear overview of what assets are available and what purpose do they fulfill, through having standard service descriptions and other information
- Service pipeline: it includes planned services, services under development for certain customer or

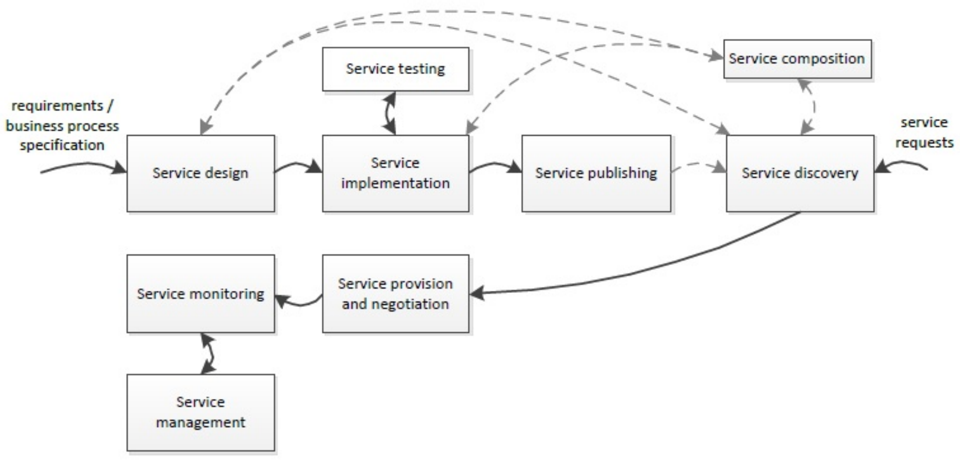


Figure 1.3: Service lifecycle

market, and potential services to be developed for future business. The pipeline offers the strategic outlook of the company's business services

- Retired services: those are the withdrawn services and no longer available to customers or to be used to assemble solutions and consumed in the runtime. A service is often thrown out when it is no longer beneficial or required

On the other hand, managing the provider's service portfolio is a discipline whereby the service provider creates, describes, and maintains the optimum set of services to offer to solve specific business problems for customers, enhance the business agility, and maintain profitability. It is considered as one part of SOA governance and an enabler to achieve business governing goals. It makes sure the provider has the right balance between investments in their IT assets with the ability to meet business needs. This includes tracking the investments and costs in services through the service lifecycle and make sure the desired return on investment and customer satisfaction in term of requirements and SLAs, are met.

As part of the portfolio management, providers should also ensure that services are clearly defined and modelled in the portfolio catalog, so they can be leveraged quickly by IT teams and business teams, but also they can be linked to business outcomes and suitable KPIs, so they get reviewed based on business needs, costs, and customer satisfaction. This enables providers to phase out services and plan for new ones based on business needs and costs periodically and only keep and manage economically viable services which are aligned with business needs and provider's strategies [28].

1.2 Motivating problem and an example business case

While software companies aim to continuously improve the efficiency of deliveries and reduce maintenance costs, managing the evolution of the service portfolio is critical for this task. With the continuous development and incremental changes in Agile, there are always granularity decisions to be taken by different development teams who only see scoped work in backlogs. Modelling assets and their usage in past solutions to maximize reusing assets and align them with business needs is crucial to optimizing development and maintenance cost as well as sustaining the portfolio. Having proper modelling technique doesn't only allow Agile teams to interpret the capabilities of assets in design loops efficiently, managing too many services that are used becomes costly and hard, which can also result into high service interruption for customer on technology changes.

A typical example from the banking sector, in which the software company Stachanov developed a platform as a set of supporting business and utility functions to help them rapidly develop web applications for *loan underwriting and credit risk assessment processes and services* (More details are in Appendix A.1). The platform has an integrated workflow engine, monitoring functionalities, a standard financial model, and other utility functions and can be included in a web solution for a certain bank as a base to implement the business functionalities for such loan underwriting and risk assessment processes. This speed up time-to-market a bit through offering basic functionalities in web applications.

As customer-base evolved, with more delivered applications and the iterative delivery of requirements to different development teams, many generic and shared business-domain functionalities ended up re-implemented in different applications or as part of custom business functionalities rather than separate functionalities and services in the platform itself to be reused across various applications. Sometimes those were also re-implemented in different components in one application due to automating similar kind of business processes in one bank for different types of loans and applicants. This resulted over time in lower productivity and higher maintenance cost as well as no significant reduction in development efforts for new solutions. It also affected the ability for introducing changes to applications without causing errors due to the tight coupling and improper modularity for business functions.

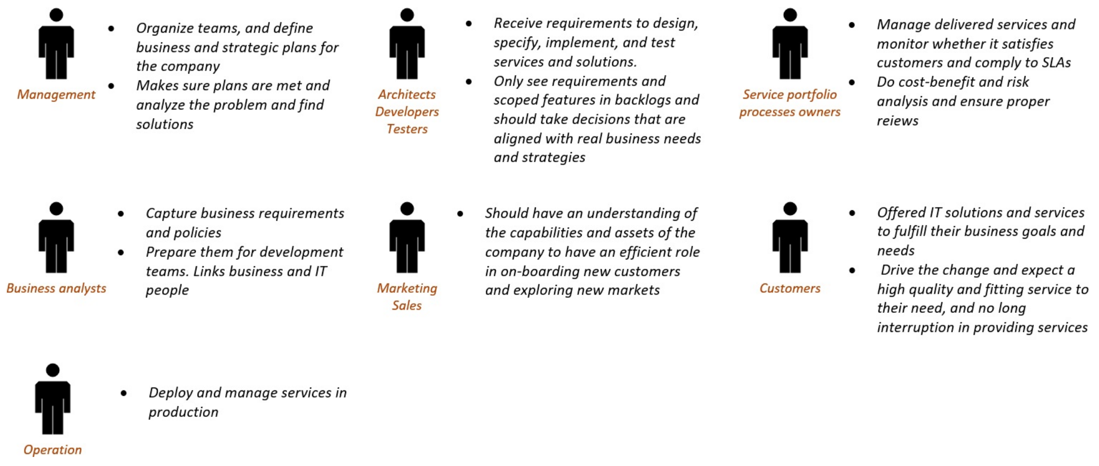
When applications evolve, not only maintenance cost can increase, even experienced team members can find it difficult to understand the capabilities of existing assets when introducing changes and adding more features due to the big responsibilities of existing functionalities and the lack of proper modelling technique for assets.

Management and business analysts at Stachanov are highly experienced in target sector needs and are aware of banking and risk assessment standards and have proper service management processes in place. Their original objective was that the platform can evolve into a set of business services to be reused due to the similarities in such business processes and targeted business domains, rather than expanding the size of solutions. However, even with competent developers and knowledgeable management & business staff, the company couldn't keep track of the impact of short term design decisions on the sustainability of their assets and expected development efforts on the evolution of assets, and couldn't evolve their assets to enhance reusability and changeability. They couldn't also connect development teams and pipeline with their service portfolio management processes to mitigate risks and take proper actions early.

The management is considering SOA adoption and developing a set of business building blocks that can be constructed and deconstructed to assemble new solutions based on the variations in banking policies, standards, and banks needs. However, they would like to adopt an effective service identification and modelling approach, so they can reduce the risks of similar problems to occur in the future and align their IT assets with business demands efficiently to optimize costs and meet their SLAs.

Due to the already big investment in the current platform and applications, critical decisions should be taken to identify potential business services from those systems. Most important that such decisions will recur in future development cycle. Taking proper design decisions and creating or expanding services while customer requirements are incrementally processed and with daily design decisions being taken by various developers is a major challenge. Therefore, we were very motivated by such a typical problem to research and develop an effective service identification and modelling approach that can enhance service identification and modelling strategies of Agile teams based on service portfolio evolution and link Agile modelling activities with portfolio management activities using proper granularity and cost indicators to receive feedback from each other and manage the evolution of the portfolio effectively.

Involved stakeholders: Since the service granularity and modelling problem is not only an architectural problem, various business and technical stakeholders are involved in the solution at any software company. Those stakeholders are presented in Fig 1.4.



Management
- *Organize teams, and define business and strategic plans for the company*
- *Makes sure plans are met and analyze the problem and find solutions*

Architects Developers Testers
- *Receive requirements to design, specify, implement, and test services and solutions.*
- *Only see requirements and scoped features in backlogs and should take decisions that are aligned with real business needs and strategies*

Service portfolio processes owners
- *Manage delivered services and monitor whether it satisfies customers and comply to SLAs*
- *Do cost-benefit and risk analysis and ensure proper reiews*

Business analysts
- *Capture business requirements and policies*
- *Prepare them for development teams. Links business and IT people*

Marketing Sales
- *Should have an understanding of the capabilities and assets of the company to have an efficient role in on-boarding new customers and exploring new markets*

Customers
- *Offered IT solutions and services to fulfill their business goals and needs*
- *Drive the change and expect a high quality and fitting service to their need, and no long interruption in providing services*

Operation
- *Deploy and manage services in production*

Figure 1.4: Stakeholders and their roles

1.3 Research project details

Proposed granularity decision making and portfolio management approach: To help companies solve the identified problem while motivating development teams to work independently and efficiently, we developed a granularity decision making approach based on software development and portfolio evolution, which integrates a collaborative modelling technique for assets and new required business functionalities in SOA solution and analyze the variations between those to discover assets for reuse in design loops.

When adopting the approach with Agile continuous development and delivery, the solution establishes and continuously maintains an up-to-date knowledge-base for existing business services and assets as well as already spent efforts to develop those services. This reliable knowledge-base makes the solution work as a self-learning feedback tool to the granularity decision pipeline to take cost-effective granularity decisions and enhance choices, as well as offers feedback the service portfolio management activities to scope work and plan for publishing services. This feedback can be also used to monitor spent costs and the impact of continuously taken granularity decisions on business in the long term. As a result those monitoring activities in the portfolio management can also update the knowledge-base based on business observations with suitable decision rules that can help Agile teams optimize their design decisions in the future before it is costly to do so.

Research questions: In this project, we aim to answer the following main question:

How to effectively manage the evolution of the service portfolio in agile software development?

To be able to answer our main question, the following supporting questions were drafted and answered during our research work:

i *How to speed up leveraging existing assets and enhance the knowledge sharing across teams?*

ii *How to continuously update the service identification and modelling strategies during the software lifecycle using the feedback from the portfolio evolution?*

iii *How to optimize service granularity with consideration to development costs and portfolio management?*

Research methodology: In this research project, we used a multi-method qualitative approach to analyze the problem and look for supporting practices in the literature to identify our proposed approach. The research methodology is presented in Fig 1.5. It includes the following steps:

1. **Systematic literature review and analysis:** we conducted an extensive systematic review of many literature sources (35+) on service identification and modelling, service granularity aspects and related work in studying the management of service portfolio evolution and the impact of bad granularity decisions on costs and assets sustainability. With this desk research, we analyzed the best practices and literature gaps and constructed our approach
2. **Development of a granularity decision making solution:** we constructed our solution by combining and enhancing best practices to fill the identified gaps in current solutions and solve the researched problem
3. **Case-study (validation and demonstration):** we demonstrate the proposed approach using a case study at the host organization

Furthermore, we also validate and discuss how our developed solution can complement or guide modern concepts in aligning IT capabilities and business needs, so it enables software companies to apply such concepts in practice.

Contributions: Our proposed solution makes particularly the following contributions:

1. It introduces a similar modelling technique for both required business functionalities in new SOA solutions and existing business services and functionalities in legacy systems and service inven-

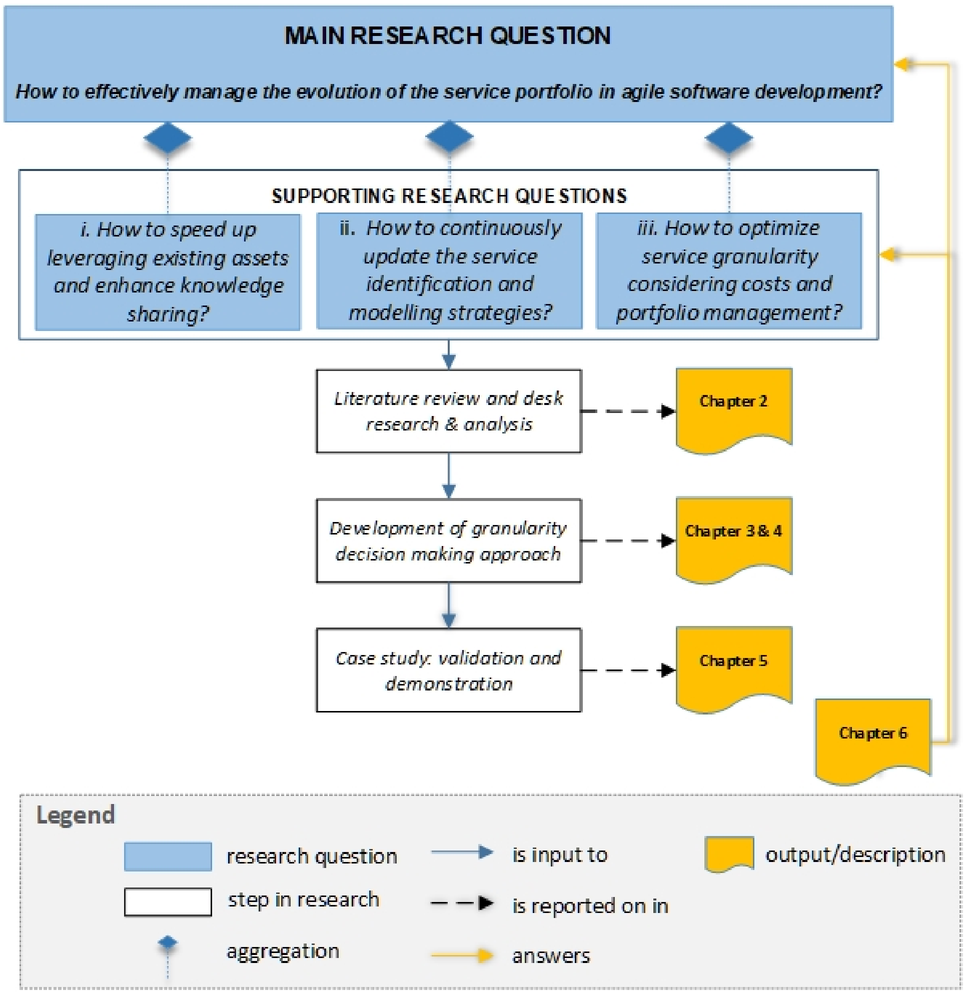

MAIN RESEARCH QUESTION
How to effectively manage the evolution of the service portfolio in agile software development?
SUPPORTING RESEARCH QUESTIONS
i. How to speed up leveraging existing assets and enhance knowledge sharing?
ii. How to continuously update the service identification and modelling strategies?
iii. How to optimize service granularity considering costs and portfolio management?
Literature review and desk research & analysis
Chapter 2
Development of granularity decision making approach
Chapter 3 & 4
Case study: validation and demonstration
Chapter 5
Chapter 6
Legend
research question
step in research
aggregation
is input to
is reported on in
answers
output/description

Figure 1.5: Research design

tories. This can speed up leveraging assets and interpret their business capabilities to optimizes assets reuse and development cost. Also, the variation analysis of required functionalities and past delivered business functionalities in the design loop prevents developers from spending unnecessary efforts and increase future costs. Moreover, it allows development teams to learn from successful cases in already designed solutions

2. It considers modelling the decomposition of existing business functionalities in legacy systems from their fine-grained function units to establish the service portfolio. Those models can be used in the variation analysis to discover potential code and reduce service implementation costs

3. It offers suitable indicators or metrics for measuring service granularity and estimate services costs to allow comparing various granularity options and reuse of existing building blocks from an economic perspective in the decision pipeline, before considering implementing new services or refactoring existing ones

4. With the continuous development in Agile projects, it works as a self-learning tool through learning from past support and models, costs and issues to take new design decisions and enhance them. This allows to sustain the service portfolio and align IT assets with business needs continuously. The self-learning dimension develops granularity patterns over time to solve similar domain problems

5. As a result of having the continuously updating knowledge-base and the integrated indicators, the approach connects development activities and stakeholders with portfolio management in a seam-

less way. It is unlike existing methods allows those to send feedback to each other and mitigate risks and coordinate between the various stakeholders around business value and customer needs

6. It can guide Agile teams in establishing and maintaining an evolutionary architecture for SOA solutions, in which change is encouraged and is revertible without causing pain to portfolio sustainability and solution's reliability

7. It is easy to extend the portfolio management and introduce more long-term indicators to measure the impact of daily decisions on business and quality of services. Those work as predictive alerts to prevent long-term damage to resource productivity, assets sustainability, customer satisfaction, and architecture resilience to change

1.4 Outline

The rest of the paper is organized as follows: Chapter 2 presents state of the art, related work, and literature gaps analysis. In Chapter 3, we present the design of our self-learning granularity decision and portfolio management approach. Chapter 4, contains the implementation details of this solution. In Chapter 5, we present a case-study from the host organization. The chapter includes a brief summary of the preliminary research activities for analyzing the problem and collecting requirements and a demonstration of the proposed solution using the case. In Chapter 6 we answer the research questions and discuss our contributions with various topics. Chapter 7, presents uses of the solution in different contexts. Finally, in Chapter 8, we share our conclusions and propose future research directions.

Chapter 2

State of the art and related work

2.1 Systematic literature review and analysis

To develop a solution that allows software companies to effectively manage the evolution of their portfolio in the long term and align assets with business demands, we reviewed various literature sources using the systematic literature review method proposed by Kitchenham [5]. The major goal was to explore best practices in service modelling and to cover the various service costs that are influenced by the choice of service granularity and service modelling techniques. Different combinations of the following keywords were used to query literature:

SOA, Service identification, Migration Legacy code, Service granularity, Business Process Management, Cost benefit analysis, Development cost, Composition cost, Decision framework, Agile, Service sizing, Reusability, Assets modelling, SOA modelling, Requirements modelling, Service catalog, Service description, Service portfolio management, Evolutionary Architecture.

The following portals and electronic databases were used to query for literature sources:
IEEE Xplore (IEEE), SpringerLink (Springer), ACM Portal (ACM), Research gate, and CiteSeerx.

To make the query outcome as reliable as possible, refereed articles in conference or journal, cited articles in other articles, publication of various academic communities, and published book with various references to were considered. We skimmed through papers, namely: *title*, *abstract*, and *conclusion*, to exclude papers not within our scope of research at an early stage. The survey led to prior research work in the concepts of agility and reusability, benefits of adopting SOA, service identification and granularity decision approaches and cost estimation approaches. Once the gathering phase was done, a processing phase has started to read papers in depth and select and categorize papers within the scope of this project. After that, a careful assessment of current work with respect to the candidate research questions was done. As a result of this, we present an extensive summary of the current work and the useful knowledge related to our research questions. The information is categorized based on the topics of interest, each category is covered in a separate section. In the last section, we briefly describe the gaps we would like to fill in this project.

2.1.1 Service granularity and service identification and modelling

Service granularity which refers to the scope of functionality in the service, has the functionality as an essential aspect by definition [13]. A service, on the other hand, should focus on a certain purpose and therefore should serve a specific functional area in the business domain. Besides functionality, Steghuis [29] identifies composability, genericity, context-independence, complexity, sourcing, and performance, as important aspects of service granularity.

To identify and create services to fit business needs and cut development costs, the literature suggests top-down, bottom up, and a range of hybrid approaches.

Bottom-up approaches: In bottom-up approaches, the service identification starts from analyzing existing and legacy systems to identify function units and code that can be reusable and provide lower cost solutions to implementing underlying service functionalities that support customer solutions. This is especially beneficial if the existing code is costly to rewrite. It can include data flow and code analysis and code dependency analysis, database reverse engineering, documentation analysis, and expert judgment in business to align extracted assets with business demands.

Kulkarni and Dwivedi [21] applied a bottom-up approach to a case in the financial industry to analyze the impact of service granularity on the realization of SOA. They focused on finding the optimal granularity in the service identification phase by starting from the case and applying function decomposition techniques to the existing system to extract the finest functional units. They used the decomposition result to form a picture over the future required functionalities and specifically focused on reducing dependencies between services to reorganize units and identify potential services, which are divided into 8 classes based on stakeholders taxonomy and identified business domain such as taxation and retirement plan.

Even though they propose to apply their approach of decomposing units based on dependencies and classifying them iteratively to arrive at an optimal service composition, their method doesn't evaluate the impact of granularity on the development cost in each iteration. Moreover, such approach can also be challenging, especially with the lack of an up-to-date knowledge-base or design documents.

In general, what is good about the bottom-up approach is that it can provide an understanding of the capabilities of current systems and offer high flexibility in composing services from existing functions. However, it doesn't necessary catch the business intent of identified functionalities and can tie services to their originating technology and environment. Therefore, it is often combined with service goal modelling, domain decomposition, and domain expert analysis to define the required business functionalities in business models and map them to existing function units to enhance the alignment between business demands and existing software assets [3].

Top-down approaches: In the top-down approach services are identified based on analyzing the application domain and the business process to build a conceptual model for functional requirements, then decompose those requirements into future functionalities to be exposed as services, based on their business value to customer or namely, their value in fulfilling a customer need or support a business use-case [15]. Also, this approach requires an understanding for the business models and careful analysis of customer demands, as well as more involvement from management, to define target service portfolios. It also requires a high level of maturity in adopting SOA and the various involved processes of creating and deploying services, and in maintaining service portfolios.

Jiang et al. [16] constructed and applied a similar approach to support the choice of service granularity based on a high-level abstraction of business process. They analyzed and decomposed various business processes for customers into smaller tasks and used the resulting statistical and similarity information to determine the service granularity levels. However, they didn't consider analyzing existing services and didn't reach the implementation level, which can result into introducing redundancies and facing difficulties in aligning the business goals with implementation domain and the functional capabilities and limitations.

In general, applying a top-down approach allows catching the business intends behind the services. However, it requires analyzing various business models and decompose them to spot recurring tasks and granular units of work, before creating those services. Otherwise, adopting it alone in the development cycle when a new solution for certain customer has to be built can lead to missing the change of reusing existing units or already available services and can increase the chance of service duplication and overlaps in functionalities which leads to a higher maintenance cost.

Hybrid approaches: To consider customer's requirements and business objectives as well as the existing systems and services, various hybrid and meet-in-the-middle approaches that combine activities from top-down, bottom-up, business goal analysis, or service goal modelling methods are used.

Those approaches often decompose application domain and business processes into granular tasks and further into logical units of work, and analyze existing code to extract function units and reorganize them based on their value to business, then they use a gap-analysis between the outcome of the requirements decomposition and existing assets analysis, driven by business goal analysis and other methods such as business expert opinion or algorithms that can reduce dependencies between services. This gap analysis usually aims to check what infrastructure, resources, or legacy code reusability in creating the identified services and how to layer services and define implementation components in one SOA solution.

Arsanjani et al. from IBM [3] introduced SOMA approach which includes, top-down, i.e. domain decomposition, goal service modelling, and bottom-up activities, i.e.analyzing existing assets. They recommend to start by analyzing SOA solution goals and extract subgoals that must be met to realize goals. They suggest identifying Key Performance Indicators (KPIs) to evaluate whether a goal can be achieved when slicing the functionalities, such as *Increasing the number of online users of banking services* when developing a solution as a set of banking services for a certain bank. On the other hand, they model business processes and partition those models into distinct domains and functionalities iteratively and perform variation analysis on required functionalities. They also analyze existing code and focus on assets that play a role in implementing part of the required functionalities. They finally suggest mapping the outcome of the source code and existing APIs analysis to the identified business functions from the top-down approach. Initially, a coarse-grained mapping of business activities to the existing assets and interfaces is done and this is further refined to increase the chance of using existing assets. SOMA offers a good degree of prescription, as the seven described activities are detailed and have an identified input and output. However, the method doesn't describe how created services can be reused in new solutions and how to maintain the sustainability of service portfolios when more SOA solutions are delivered.

Albani et al. [2] discussed the granularity from a functional point of view. They proposed a method to identify existing functional components in current systems and map them to the required business tasks. They suggest grouping rules that are based on maximizing coherence and minimizing dependencies between functional components and to have components that belong to a functional perspective grouped together.

Other proposed methodologies provide general recommendations for combining bottom-up and top-down activities, such as SOAD (Zimmermann 2004), SOUP (Mittal 2006) and SOAF (Erradi et al 2006). However, they miss identifying how to match needed services to existing services for new solutions and don't discuss how to sustain the service portfolio. On the other side, many other methods focus on the technical perspective of creating services and use algorithms to find optimal granularity and composition of IT services to deliver functionality by reducing dependencies or optimizing performance and cohesion. Those methods usually focus on reusing legacy code and migrating systems to SOA, rather than on developing solutions for customers in a later stage of adopting SOA and service lifecycle (Offermann 2008, Winter 2008, Verma & Shah 2011, Jiang et al 2011).

2.1.2 Modelling techniques

Services and software assets: Modelling functional requirements and align customer business goals and IT capabilities are often challenging. When transforming customer requirements into software or services, the business process and the flow of activities in requirements are captured and refined. There are various modeling notions that can be utilized to model the various activities and the flow of work. One of those modelling notions is the Business Process Model and Notation (BPMN), which offers a strong tool to model the flow or orchestration of high-level business services to realize customer requirement and gives an abstract view of how business activities take place by customers and what dependencies between those exist [4]. Statecharts and petri-nets to analyze and validate the sequence of execution in composite services are also used [18]. The underlying functionalities of the high-level business tasks or services are usually defined by breaking down user stories in Agile into manageable stories, so they can be worked on iteratively. Those functionalities are often modeled with Unified Modeling Language (UML) (activity version) or call hierarchy diagrams and feature trees [34].

Besides modelling requirements and service composition when designing new SOA solutions, there are also many modelling languages and ontology proposed in literature (SOMA, SOAF, UML-s, SoaML, OASIS) that offer common semantics frameworks to describe service assets and also the relationship between the various elements in SOA such as service composition relations, and to clear the ambiguity across different applications and ease discovering assets by architects and developers to be reused in a new solution. On the other hand, for service inventories and existing services, since companies often reuse IT services to assemble various solutions and realize different business services, literature emphasize the need to have a standard knowledge model to describe services and understand the composition of business services and solutions, so SOA services can be reused when possible [33, 3].

Architectural decisions: Modelling information about architecture decisions such as granularity decisions is important according to Zimmermann [24]. Zimmermann constructed an architectural decision modelling guidance and identified decisions by a set of informational representations. A decision according to him should be triggered by a problem statement or an observation. The information should include the recommended decision or action and decision drivers. Decision drivers can be mainly, quality attributes, cost attributes, measurable business benefits, or feasibility of using technology. Furthermore, Zimmermann suggests including architectural alternatives of the decision to not get stuck within one option and miss the opportunity to consider and compare other options or use them in the future to guide and optimize the decision making.

Even though Zimmermann talks about technology, tooling, and high-level architectural and outsourcing choices such as workflow and messaging engines, his proposal of logging problems and linking them to risks seems suitable in the context of our research project.

2.1.3 Service granularity and costs

The granularity of SOA services is crucial for development costs. It can affect reusability while increasing complexity and change cost and lowering coherence [29]. Development cost in SOA applications doesn't only come from implementing new services that are needed or not discovered in the service catalog, considering reusing existing services by combining those with other services comes with composition costs. Having too many fine-grained services can increase reusability of those to realize various business processes and reduce implementation and maintenance efforts, while having coarse-grained services with minimum dependencies on each others helps realize the required business functionality often with less SOA services, thus less composition efforts [16, 21]. Literature suggests that the more coarse-grained the services are, the fewer composition efforts may be required because of having fewer components or services to combine to realize a functionality. Fine-grained services in principle can be more reusable that coarse-grained services. However, this is also related to the type of functionality and its business value [13].

Besides that, service sourcing options should also be considered [29]. With open-source solutions or outsourcing part of the work trending, discovering services in repositories or lifting part of business services outside the company through strategic partnerships became very common. This made choosing the granularity of business services critical to the integration efforts and enhancing the chances of integrating external services in solutions to fulfill customer needs. When discovering and integrating services from the local service catalog, public repositories, or partner services, having a standard description of service interfaces for internal services is important to make the discovery and integration process efficient and doesn't miss a chance of integrating external services and cutting costs. Cardoso et al. [14] also concluded that granularity can affect the seamless integration and plays an important role in aligning IT services with business needs, especially as an IT service can be reused across multiple business processes to realize different business functions.

Moreover, change cost when requirements change and solution evolves should also be considered when slicing business functionalities and considering levels of granularity, but it is hard to predict beforehand. According to Zhan et al [34], change cost increases when the responsibility of services increases and vice-versa. In other words, the service is coarse-grained service, the more efforts may be required to make a change to it or refactor it. On the other hand, a change in service or technology and redeployment of services that are reused and heavily consumed can affect business more due to the service downtime.

The long term aspect is very critical as well. With the continuous development of SOA applications, introducing services with improper granularity or overlapping services can increase service inventories management and re-engineering efforts as well, which is crucial for business [28].

2.1.4 Cost estimation methods

COCOMO II is often used in component-based development with estimated number of Source Lines Of Code (SLOC) or function points to estimate costs of implementing services. However, Tansey and Stroulia [30] attempted to use this method to estimate the cost of implementing and migrating systems into services. They concluded that the method should be extended to consider the composition costs and reusability benefits. Also, Santillo [27] tried to use function points in estimating service implementation cost. However, since functions in SOA represents a real-life encapsulated business service to customers, he concluded that this algorithm is difficult to apply and that the effort to consider existing implementation (e.g. legacy code) as a service cannot be estimated. He alternatively suggested measuring costs on the service level. This was also proposed by Liu et al [17], who suggested to estimate the service size and combine it with infrastructure and technology cost. However, they didn't present how service size is measured. To look further into how sizing services can be done, we researched the metrics proposed by Zhan et al and Zimmermann [34, 23] to quantify the functional scope of a functional component in respect to the business functionality scope in a solution and we improved it to be used in the context of the service granularity.

Furthermore, expert judgment is also a common method to estimate the cost. However, since it's driven by experience in certain projects, it is hard to generalize and can easily fall into the trap of underestimation due to human biases, SOA complexity, and the uncertainty about choosing right modularity and granularity, in respect to future reusability and costs as well as customer change of demands. Therefore, it is recommended in combination with other methods such as analyzing historical data or calibrating cost factors periodically to cover past efforts.

In Agile projects, cost or efforts estimation is often done by extrapolation methods using historical cost data and backlogs, which takes into consideration the productivity of teams and the development work done is in respect to the required development efforts to have a final product. User stories are analyzed and broken down to identify functionalities. Therefore, story points are often used instead of SLOC as a measure for size and complexity. This is because project scope and sprint scope don't map easily to the decomposition of functionalities needed for function points, especially when function units may overlap many stories and services can be combined in many ways to implement a wider business function and high-level features [10].

2.1.5 Risk mitigation

Risks discussed in literature concern the granularity and service design decisions are at the technical level, such as producing services with shared data models that requires revisiting the implementation of other services when one service changes, and other risks concerning service performance and business disruption due to IT service downtime or incidents [9, 31].

Other risks concern sourcing and managing knowledge about existing assets and delivered SOA solutions, as IT team members can resign or undergo bad conditions which disallow them from collaborating in labor. Therefore, having standard procedures to manage services and a standard knowledge model for available services and assets are two important aspects to do the required development task, manage the level of service quality efficiently, and mitigate such risks [36].

Moreover, without a periodic assessment of the benefits of available and managed services as well as reflect on weaknesses and strength of design decisions when solutions evolve, it is difficult to keep solutions sustainable and service inventories manageable. Improper design decisions can lead to having services with small business value or services that cost a lot of efforts and customer's business disruption when technology changes. It can also make it hard for the software company to assign and evaluate proper KPIs for assessing the risks, costs, and benefits of delivered business services to customers [28].

2.1.6 Assets sustainability and Evolutionary Architecture

In software engineering, sustainability is defined by the ability of software to preserve over time. Choosing improper boundaries between services in a SOA solution can result in higher change cost and future maintenance and re-engineering costs. This can affect sustainability negatively and can cause producing complex or unnecessary services and a growth in size and number of services with low reusability and is costly to fix later[26].

Sustainability is crucial in Agile continuous development and delivery as well. There is always the need to add extra features and perform changes rapidly without breaking the solution [6]. Leveraging service assets that are coherently described can increase the chance of reuse instead of creating new (possibly overlapping) services to assemble the scope features. This way, the incremental development helps in optimizing the choices to slice functionalities and development costs iteratively to meet customer needs basing on decisions in past releases and existing assets. Therefore, it helps in maintaining service portfolios and suitable service assets for business [11].

Besides that, it is common when adopting SOA to create generic service assets to serve business for years and to identify services in a solution that can evolve following the incremental development and changes in requirements without spending too much efforts or affecting solution sustainability.

Researchers tried hard to find an answer to the question: how is long term planning possible when things change unexpectedly. They tried to observe the software architecture and assets evolve in various projects over time to be able to define proper strategies to reduce creating solutions with high future maintenance costs and are non-resilient to change and adds more service assets that can increase the size of service inventories and are hard to be reused. However, this old method of solving the problem of uncertainty turned to be ineffective with the continuous changes in ecosystems, as changes are hard to predict [25].

As an alternative to such strategies and to enable development teams build solutions iteratively, facilitate changes, and keep the software reliable every sprint, the evolutionary architecture idea came to life which is suggested to ease change and rolling back changes and facilitates experimentation and response to customer feedback without having a damaging impact on solution sustainability and increase in maintenance and re-engineering costs. The domain perspective of the evolutionary architecture which is important in modern design is inspired by Domain-Driven Design (DDD) to support the incremental guided change in Agile as one dimension between multiple dimensions. Changes are suggested to be isolated based business contexts, i.e. business domains, so teams can be organized around domain products. Other key characteristics and principles in the evolutionary architecture are as follows:

- Modularity and coupling: the ability to separate the system into components with clear boundaries and minimize dependencies to facilitate change and revert change with less pain and to have a reliable working product each iteration
- Separate domains: assets should be organized per domain and around business capabilities to reduce coupling, which moves SOA from using partitions of services in technical layers, towards partitioning services based on business needs
- Fitness functions to guide change: those can be used to measure whether service assets fulfill non-functional requirements and the evolvability of architecture and solutions
- Last Responsible Moment: decisions should be delayed until enough information is available. It would be efficient to guess right what the world would be like in 15 years from now. Unfortunately, systems don't display the same characteristics and demands can change. Therefore, it is important to know when to make a decision to change and evolve assets
- Bring the pain forward: with frequent releases, the development pipeline should allow for fast roll-out and rollback. It may hurt to deploy frequently, but this allows for continuous improvement and lower efforts over time. This way, developers can learn how to make their implementation more integratable and improve loose-coupling
- Experimentation and technology change: facilitate experimenting with various options and new technologies which allows for architecture to evolve and change without breaking other components

Evolutionary architecture however is still an abstract idea. There are no concrete practices and activities that can be followed to establish and maintain such an architecture for SOA solutions.

2.1.7 Service Portfolio Management

Traditional management approaches identify one central business representative to control the various processes and management of the business services and the quality of services the company offers to its customer. This central governing body usually reviews planning new services and changes before they are implemented. It can hardly address collaborative decisions and the outsourcing part of work. Also, such processes usually focus on faults and delivery efficiency rather than assets capabilities, lifecycle, and evolution. This results into assembling assets that are designed for success in one solution rather than being resilient to change and reuse.

Modern organizational structure on the other hand focus on giving autonomy IT and business teams and enhance the coordination between them through standardization of flow of work, training, and knowledge [20]. All teams are responsible for making the story successful. They usually have clear responsibilities and should access the right information at the right time to be effective in performing their job. Moreover, With the modern iterative delivery, outsourcing, and customer experience based approaches, standards such as ISO/IEC 20000, ITIL, TOGAF, and FitSM are proposed to offer service providers models, templates, and procedure to enable them have collaborative management of service portfolios and meet SLAs with customers. It allows supporting customers and follow up on changes and fixes to delivered services efficiently through structured processes and clear processes ownership [28, 12].

Compared to other standards, FitSM is considered to be applicable with less efforts. It offers lightweight processes for service portfolio management, team management, and risk management, among others. It is deemed to allow service providers to govern their delivered business services and resource investment, and coordinate roles and responsibilities of IT and business stakeholders around supporting customers needs. This is especially important when business grows and more services need to be managed.

2.2 Literature gaps analysis

In general, hybrid approaches that offer strategies to decompose requirements, extract existing assets and analyze the gap between them can increase the opportunities of reusing an existing component or implementation and optimize the alignment of existing IT assets and requirements in a new SOA solution. Approaches such as SOMA and SOAF provide detailed activities in the service identification and modelling phase and combine business process and domain decomposition with business goal analysis and existing code analysis to identify services. However, they don't offer an evolutionary overview for service portfolio when solutions evolve and more solutions are deployed. Those approaches don't discuss various options in taking granularity decisions and include the development cost dimension. They also don't offer a methodology to allow IT teams who only see backlogs and scoped features leveraging available assets and already delivered solutions efficiently and understand how those services were used in past and can be used to assemble business services and solutions, so they can reuse them to cut development costs.

Without proper modelling strategies to take into consideration the evolution of assets and speed up leveraging existing assets and understanding their capabilities to be reused, Agile teams may take improper granularity decisions in the design loop and spend efforts on creating services that may be hardly reused in the future and increase the size of service inventories. The complexity of leveraging and managing a large number of assets which will be high can cause a reduction in resources productivity and efficiency of deliveries. Moreover, the re-engineering efforts to fix overlapping services, merge services, or phase them out while solutions in production still use them become very high and costly. This can also cause unnecessary business disruption to customers.

On the other hand, service identification methods that use local optima algorithms to slice existing code to reach acceptable dependencies between services and combine it with requirements may increase complexity and doesn't consider portfolio evolution as well. They don't necessarily align services with the business needs and can generate services that are tied to their technology. When services and business needs start to change, optimum loose-coupling and creating replaceable services sound to be a dream rather than an achievable goal. Keeping track of solutions costs and portfolio sustainability and the status of service inventories while daily design decisions are being taken by various developers working on different projects stays hard.

Even in the example of the host organization, when web applications with many similarities are being developed and with very knowledgeable business people in market and competent development teams, the time pressure from customers and with the lack of visibility and management of the existing assets companies can fail in transforming their business strategies and business portfolios to suitable service assets, and can introduce assets that need more efforts to be integrated in new solutions and to be maintained.

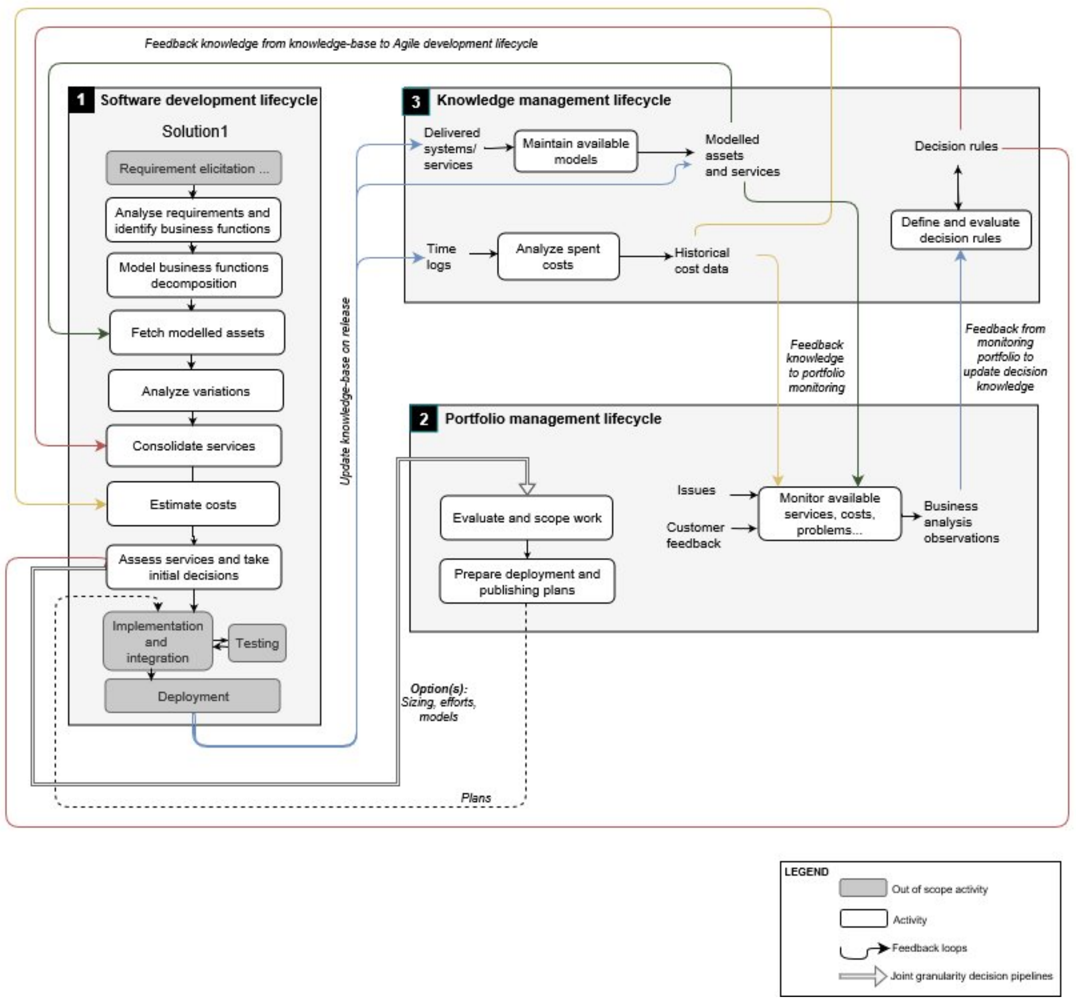
As a conclusion, when solutions evolve and with scattered knowledge about the business capabilities of existing assets, past development efforts, and the impact of the continuously taken design decisions on costs and business, it is hard to stay content with the evolution of assets and maintain a set of assets that suit business needs and are affordable to manage. From our literature review, we couldn't find an effective approach for slicing functional scope and identifying and managing required services that take into consideration those aspects.

To fulfill the identified gaps in existing approaches, we integrated best service identification practices and added the development cost dimension and a knowledge-base to reflect current portfolio situation in one approach (See Chapter 3).

Chapter 3

Granularity decisions based on software development and portfolio evolution

In this chapter, we present the core idea of our approach for identifying and modelling services for new solutions in Agile development methods and managing the evolution of the service portfolio. We explain the various cycles in the approach and how they give feedback to each other with the continuous development and delivery. Fig 3.1 depicts the basic idea and the loops in the approach.



Feedback knowledge from knowledge-base to Agile development lifecycle
1 Software development lifecycle
Solution1
Requirement elicitation ...
Analyse requirements and identify business functions
Model business functions decomposition
Fetch modelled assets
Analyze variations
Consolidate services
Estimate costs
Assess services and take initial decisions
Implementation and integration
Testing
Deployment
Update knowledge-base on release
3 Knowledge management lifecycle
Delivered systems/ services
Maintain available models
Modelled assets and services
Decision rules
Define and evaluate decision rules
Time logs
Analyze spent costs
Historical cost data
Feedback knowledge to portfolio monitoring
Feedback from monitoring portfolio to update decision knowledge
2 Portfolio management lifecycle
Evaluate and scope work
Prepare deployment and publishing plans
Issues
Customer feedback
Monitor available services, costs, problems...
Business analysis observations
Option(s): Sizing, efforts, models
Plans
LEGEND
Out of scope activity
Activity
Feedback loops
Joint granularity decision pipelines

Figure 3.1: Integrated cycles in proposed approach

3.1 A general idea

The activities in our approach can be split into three cycles:

- Software development cycle: this cycle includes the activities to decompose and model requirements and access models for existing assets and services, to identify and model required services in the solution. The activities can be initially used to establish the first portfolio of services from existing legacy systems and envisioned requirements for future customers and to migrate legacy systems to SOA services. In such case, assets should be analyzed and modelled

- Service portfolio management cycle: this cycle include the assessment of the identified and modelled services from the development cycle. Also, it covers scoping and planning work in the portfolio pipeline. Besides that, in this cycle the software company monitors deployed services based on analyzing planned investment, already spent efforts, raised issues, customer feedback, and other observations by business and IT stakeholders to take action and enhance future decisions in the granularity decision pipeline

- Knowledge management cycle: the activities in this cycle maintains the supporting knowledge about delivered business solutions to customers that is used to enhance the granularity decisions in the decision pipeline. The knowledge includes past spent efforts on developing services and SOA solutions and proper models for available business services from service assets to speed up leveraging them and enhance assets reusability

3.2 Detailed explanation of cycles

3.2.1 Software development cycle in Agile development

Those activities concern taking granularity decisions to identify and model services starting from requirements and available assets and assembled business services. The activities are as follows:

Analyze requirements and identify business functionalities: This step involves a top-down approach starting from functional requirements and user stories. Those can be broken down into business processes and high-level activities. Each activity can be decomposed further to extract the various business functions and their underlying functionalities.

Model business functions decomposition: The decomposition of required business functions should be modelled, so it can be compared to existing business functions and services in already delivered SOA solutions. This decomposition is the base to slice the required scope of functionality and define levels of granularity for services based on leveraging existing assets and integrated cost metrics.

Extract and model existing assets: To use existing SOA services and already developed business functionalities in current solutions and cut development cost, the solution considers to extract and model similar business functionalities to be compared to requirements.

At an early stage of adopting SOA, companies should start from already delivered or legacy systems. Function units that contribute in realizing similar business functions or can help in realizing part of those should be extracted. Their natural decomposition in delivered solutions can be modeled using similar modelling technique as requirements to ease comparing both models and reuse of existing assets to reduce costs and not increase size of service inventories with overlapping assets.

In the next iteration of the development cycle, business services and solutions that are already developed and deployed will be already modelled from existing IT services and can be fetched from the knowledge-base instead.

Analyze variations: Variations between the different required business functions and also with extracted/fetched precious assets can be analyzed to spot deviation levels and similarities and avoid creating overlapping and redundant services. This can help take granularity decisions and maximize reusing existing legacy code and existing (composite or atomic) SOA services.

Consolidate services: As a result of the variation analysis, there will be multiple possibilities to define levels of granularity for the required functions and to extend the portfolio in case of legacy to include more functionalities from assets. As a result, a list of candidate services and their granularity for every option should be identified to be assessed from an economic perspective in next steps and take final decisions. There may be a need to readjust requirements decomposition or refactor existing assets to maximize the reusability, as one of many options. When adopting the solution for some time, decision rules can be derived based on following up on issues, analyzing past spent efforts, and other metrics to measure the impact of past decisions on business and costs.

Estimate costs (efforts): In this step, costs for creating and composing services and for refactoring existing code and services in the identified portfolio should be calculated. The approach integrates the use of historical cost data from already created services to have a more accurate cost estimation. It is important to log spent efforts in past releases and to size services and their functional scope in able to fetch proper past data and take cost-effective granularity decisions. When having various options or when starting from different models in legacy code or when decisions should be taken to reorganize business functions composition from existing function units, there is a need to estimate the cost for all options to take final decisions.

Assess services and take final decisions: Based on the estimated costs and identified levels of granularity of services, team lead or portfolio process owner should take final decisions based on costs and discovered options to plan services in the service pipeline and allocate resources. Currently, the solution only considers the cost factor and the capabilities of leveraged assets to take decisions. However, in the future, it is easy to add other long-term indicators in the portfolio management cycle to derive observations and enhance the decision support.

3.2.2 Portfolio management

A part of the portfolio management is to guarantee that changes and incidents in services and solutions that are deployed are managed as well as evaluating existing IT services and their suitability for business and customer needs. It is important in SOA to continuously monitor the quality of offered services to customers and costs and coordinate between teams towards supporting customers and their changing requirements through proper processes. This is also important to maintain a continuous balance between short-term decisions that optimize local development costs, namely, efforts in one development iteration, and the long term impact of taken decisions on costs and business.

Plan work and deployment: As part of the portfolio management to plan for creating, deploying and publishing services and to allocate proper time slots and resources. The estimated efforts and measured granularity can be used to scope development work. Moreover, the modelled services can help in planning for deployment.

Monitor deployed services and costs: In case of change requests or incidents, those usually go through a standard process with dedicated process owners and participants so process owner can report to them to follow up. This is key for delivering high quality solutions to customers and meeting service level agreements. In this regard, templates and processes in standards such as FitSM can be used to complement our methodology and make sure changes and fixes are handled by development teams efficiently. This is also important to keep customers informed about the status and to have their feedback reach the right teams. However, this requires up-to-date information about assets and spent costs, which is taken case of by the knowledge management cycle to make sure that the knowledge-base reflects the current situation and the evolution of the portfolio. The logged costs from each development iteration can be used to monitor investments and compare it to the initial estimated costs and sizing as well. This is because the development teams will work on the identified services iteratively and each iteration, actual efforts/costs are logged. All those monitoring processes offer an up-to-date feedback to solution architects to revisit published decisions and decision rules, which allows to enhance granularity choices in next development iterations.

3.2.3 Knowledge management

Adopting the proposed modelling and decision making approach for every development iteration and for the various products establishes and maintains an up-to-date knowledge-base, which contains the models of already delivered business services and functionalities in previous releases as well as past spent efforts. However, it is important to schedule periodic checks and manage this knowledge to make sure that it is reliable and standardized, so all teams can access and use it efficiently. Interpreting the feedback from knowledge-base to the development pipeline correctly and quickly is important to avoid introducing improper granularity decisions and create unnecessary services. Moreover, modelled business functionalities and assets can be assessed periodically by solution architects and when recurring problems exist, to check overlapping capabilities and discover new potential business services based on the variations.

Interpreting the knowledge correctly is also important for the activities in the portfolio management pipeline to monitor costs and issues. It also helps planning change requests and bug fixes and enhances communication with customers to support them efficiently.

Maintain available services and modelled assets: When starting from existing and legacy systems to establish the initial portfolio of services, those models will be generated from existing assets and be published in the knowledge-base. When using our service identification and modelling activities in every development cycle, identified and delivered services will be modelled and added to the knowledge-base. However, there is a need to keep maintaining the modelled business services in the knowledge-base due to the evolution of solutions and phasing out services.

Maintain historical cost (efforts) data: Spent costs on already created and deployed services should be analyzed to maintain a standard historical cost data for estimating costs in the decision pipeline. It is important to keep an up-to-date overview for both: 1) supporting the decision making process; 2) and evaluating the impact of those on costs and business in the portfolio management part. This task of maintaining an overview of historical cost data that reflects the current situation should be assigned to a dedicated person. It can also be automated in the task tracking system, so once spent efforts are logged, the overview is updated in real-time. However, this requires good discipline from development teams.

The historical data is important to form a consistent and realistic cost estimation, take proper design decisions, and manage service pipeline and resources. It can also help service providers to understand the initial investment on services for periodic financial and cost-benefit evaluations.

Derive observations and decision rules: At a later stage of adopting the solution and with the continuous delivery, observations can be derived from recurring problems, customer feedback, and analyzing spent costs in previous releases, among others, to support choosing levels of granularity in the granularity decision pipeline. They can also be found during the variation analysis and service consolidations, so improper past granularity decisions for similar functionalities can be reported and published.

Observations can be used to define decision rules that support choosing levels of granularity in the future. Rules can also be derived from business strategies, best practices, market standards, and reuse plans for existing services. They can also reflect success stories and reference models in certain domain and industry to support SOA designers and development teams.

Introducing such rules allow developing granularity patterns for solving similar domain problems and enhance the learning process based on problems in past solutions. However, they should be revisited periodically to enhance them based on the current situation of costs and assets. Besides that, solution architects can also schedule periodic checks to analyze existing business services in the portfolio and spot deviation levels and redundancies or overlapping services to publish such rules as supporting knowledge in the decision pipeline.

A decision rule can be identified by an observation. This observation can either a problem or a situation in which certain functionality need to be realized and granularity choices should be taken. Rules can also concern a certain project, a set of projects, or certain domain or subdomain and can be published by responsible solution architects to be queried in design loops. Each rule should have: a recommended action and possibly drivers, a reference to a good or bad granularity example from previous solutions, risks (reflecting a learned lesson and found issues), and optionally alternatives (see Fig 3.2). A simple example of a decision rule is as follows:

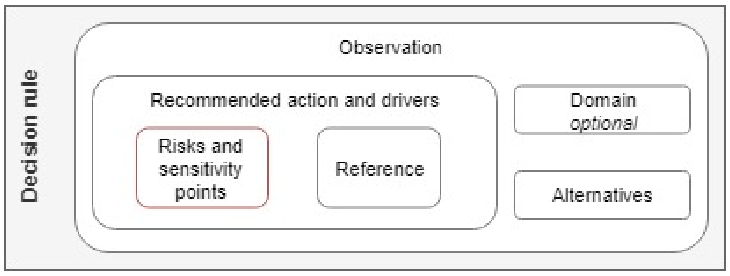


Figure 3.2: Decision rule

OBSERVATION
A credit scoring model is based on business risk ratios

Recommended action and drivers: Business risk ratios are usually standard ratios that are calculated for various scoring and risks assessment models. Implement the scoring model and ratios in separate services

- **Risks and sensitivity points:** Including ratios functionalities to credit scoring model makes them hard to be reused to assemble various models and business risk features
- **Reference:** a reference to an example

Alternatives: Refactor existing service or functionality and separate the scoring model from ratios

Introducing decision rules at a later stage of adopting SOA don't only support service decisions in the design loop, they work as a lightweight decisions logger and alignment of all teams at the company.

3.3 Interactions between lifecycles through the knowledge-base

With the continuous development and delivery in Agile, every time granularity decisions should be taken, the knowledge in knowledge-base for modelled business services and functionalities and spent efforts as well as past issues and risks is fed back to the design loop to take cost-effective granularity decisions and sustain solutions and portfolios (Fig 3.3 shows the joint decision pipeline and feedback loop).

Thanks to the continuous delivery and the knowledge management cycle, the maintained knowledge always reflects the current situation to give a reliable support to designers and portfolio staff taking proper decisions. On the other side, observations and problems in the portfolio management cycle are reported back to solution architects to update the supporting rules and enhance decisions. Therefore, the evolution of service assets in the portfolio through the continuous development and delivery is managed continuously through the continuous learning from past deliveries and granularity and cost indicators.

Connecting Agile development, portfolio management, and knowledge-management cycles through the continuous loop in one approach allows Agile development and portfolio management cycles to interact and send feedback to each other, rather than having them separately. Those are usually managed by different types of stakeholders who see different pieces of information and have to do their job. While in our approach the knowledge-base establishes the link between those cycles. Integrating all activities in one solution therefore allows to mitigate the risks of increasing future costs and size of inventories as well as lowering portfolio sustainability.

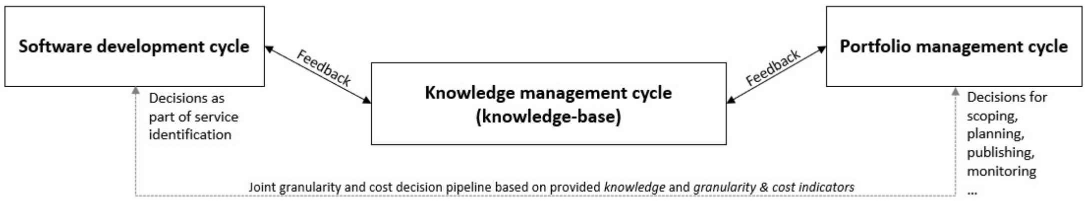

Software development cycle
Feedback
Knowledge management cycle (knowledge-base)
Feedback
Portfolio management cycle
Decisions as part of service identification
Decisions for scoping, planning, publishing, monitoring ...
Joint granularity and cost decision pipeline based on provided knowledge and granularity & cost indicators

Figure 3.3: Interactions between cycles

Chapter 4

Approach implementation

In this chapter, we describe the implementation of our proposed approach for slicing business functionalities into services when designing new solutions and for managing and monitoring service portfolios.

The approach aims to establish and continuously maintain supporting knowledge about assets and past design decisions and efforts for taking collaborative and cost-effective granularity decisions without affecting future costs and both solutions and service inventories sustainability. It works as an integrator for both Agile development activities and portfolio management during the evolution of solutions and service portfolios in Agile.

Before we present the detailed activities, we would like to define important strategies in the approach.

4.1 Related definitions

4.1.1 Business functions classification

To speed up leveraging existing services when designing a new solution, and to organize delivered services based on business lines and domains, so stakeholders can be organized around same domains, we propose classifying business services and required business functions following two main classes and one supporting class.

- **Business-dependent service:** a service which contains functionality that can be reused in solutions targeting a certain business domain or subdomain
- **Generic service:** a service which is independent of the business domain and any business specific context, e.g. email notification service, or generic document generation service. Those can be often implemented using Commercial off-the-shelf (COTS). Even if implemented inhouse, their evolvability is low. They can be highly reused in multiple solutions and in multiple business processes with no dependency on other components rather than integrated COTS
- **Supporting service:** a service is needed for a business function, but it doesn't cover a core functionality and it is not a differentiator for the business domain

A subdomain usually corresponds to a different part of the business. In loan origination domain, *risk assessment* and *customer management* are two different subdomains or function areas and are usually managed by different departments and end-users of different roles in the bank. The first one may also involve different business experts, business analysts, and development teams working on similar solutions at the software company that offers those solutions. Business subdomains can be decomposed further into other subdomains or functional areas, depending on the type of solutions and functionalities. For example, *account management* can have *customer account management* and *transitions and payment management* as two subdomains. Each of them may include various services and business functions.

Separating business services based on the type of business functionality or in other words, the function area they cover in SOA can accelerate leveraging legacy code and similar delivered business services when decomposing requirements for a new SOA solution. This can support a simple one-to-one matching of existing services and required services in the development cycle, and in finding suitable service assets to be reused to realize required functions which were used in past solutions [21]. This also allows to organize teams around subdomains and function areas, so they can develop granularity patterns and enhance decisions with the continuous development and delivery and incidents analysis. Business analysts, marketing, and portfolio staff can also be organized in a similar structure, which is recommended to promote the continuity of teams and the coordination to deliver SOA applications [20, 36, 28].

4.1.2 Modelling technique

We propose to use a typical directed graph for modelling required business functions and their underlying functionalities in SOA solution, and to set the identified services on the graph to indicate the level of granularity of each service. An example of such a graph is shown in Fig 4.1. A graph will have one head node which is the high-level business function that requires execution of the following nodes to be realized. A leaf node is an atomic function that cannot be decomposed further with all nodes in a business function graph disjoint. This modeling will also be used for modeling a business function from existing units in legacy code as well as in the business service catalog.

For modeling the whole functional scope for a set of feature or a solution in the development cycle, we will use similar graphs. Any middle or leaf node can have one or more incoming arrow from other graphs or can be repeated in different graphs, when the function unit is required to realize more than one high-level business function. This is unlike feature trees approach which doesn't allow to have more than one head node and joint nodes between different branches, in our context, between different graphs [34]. A business unit that is needed to realize various business functions can indicate the need to include the operation in service or create a service to expose it.

To identify services on graphs, a service can be drawn at one node of leaf to indicate the scope of functionality this service contains (See Fig 4.2). Moreover, in Fig 4.3, we show a simple business function that requires two other functions according to requirements. This function can be realized by s2, which will implement the following units (leaves) in the required sequence of execution. Otherwise, it the functionality can be realized by creating and orchestrating s2' & s2" services instead, which are more fine-grained than s2.

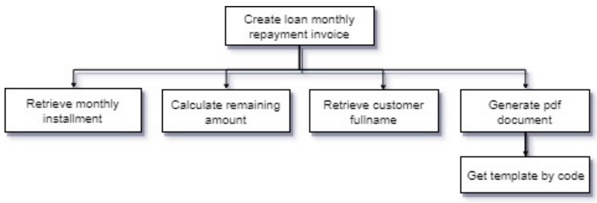

Create loan monthly repayment invoice
Retrieve monthly installment
Calculate remaining amount
Retrieve customer fullname
Generate pdf document
Get template by code

Figure 4.1: Function graph for invoice creation

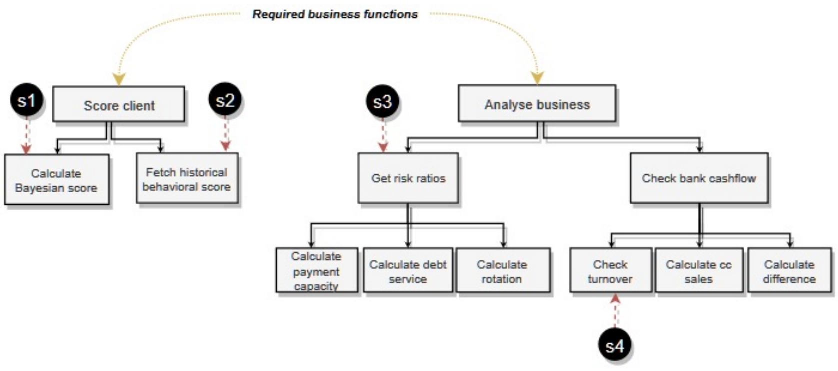


Figure 4.2: Level of granularity and service reuse for two functions

4.2 Approach implementation

In this section, we first discuss how the software company can identify and model services based on requirements and to consider legacy and existing solutions. The activities can be applied to either create the first set of services in the portfolio from existing systems or at a later stage when new SOA solutions are designed.

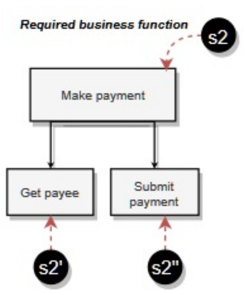


Figure 4.3: Possibilities for identifying services

Next, we define granularity and cost estimation metrics that can help in estimating services costs when taking granularity decisions in the decision pipeline.

After that, we describe how those metrics and modelled services can be used to take granularity decisions in the pipeline.

Finally, we discuss how the continuously updating knowledge-base can contribute in giving feedback to Agile teams in design loops as well as for the portfolio management activities and receive feedback from the portfolio management to enhance the knowledge that is used by Agile teams.

4.2.1 From requirements and existing systems to modelled services

To implement business functionalities and identify granularity levels of the underlying services from requirements, we start by applying a top-down approach starting from elicited requirements and business processes to decompose business domains covered by requirements and classify and decompose required business functions to model them. Then, a bottom-up approach is applied to consider existing assets. For the first time applying this approach, existing and legacy systems should be analyzed to extract precious business functions and their natural decomposition in the code to model them. Combining those two approaches allow Agile teams to consolidate services and use our decision model to identify suitable levels of granularity and assets reusability to cut costs. The flow of activities is visualized in Fig 4.4. In the following section, we describe each activity in detail, when having existing and legacy systems.

Requirements and business process analysis: Starting from requirements, the involved person in preparing requirements should identify the business processes and business domains covered. Business domains should be decomposed into distinct subdomains and function areas that will be the base for classifying business services. At an early stage of adopting this approach and delivering solutions in certain domains, there will be few classes and services under each domain and subdomain. It is the responsibility of the person who is preparing requirements for development teams to maintain classes. Those classes become the service categories in the catalog to search and compare services when designing new SOA solutions. On the other hand, the company may start from envisioned requirements to establish the first set of services in the portfolio for future customers.

For the required processes or the high-level activities, we apply a process decomposition technique to extract the various business functions in each of them. The business functions are our base for identifying candidate business services. Those business functions are still coarse-grained and should be categorized by subdomain before they are decomposed further. However, some of those business functions may not be tied to any specific function area, but rather generic.

Some business functions can be possibly decomposed further. Each of those functions may or may not require other units to be realized. For example, for a process *Manage loan payment*, a business function *"Make loan payment"* requires: *"calculate amount"*, *"get customer account"*, *"process payment"*, and *"notify customer"* functions. While *"process payment"* may also need various activities such as *"verify payment"* and *"post payment"*.

Modelling required business functionalities: We propose to model the business function decomposition using the directed graph. At the end, the scope of functionality to be realized in a solution will result into multiple graphs. Each graph has the high-level business function as a head node and following functionalities. Fig 4.5 presents this modeling. An activity *Activity1* needs 3 business functions (services) *f1, f2, f3* to be realized.

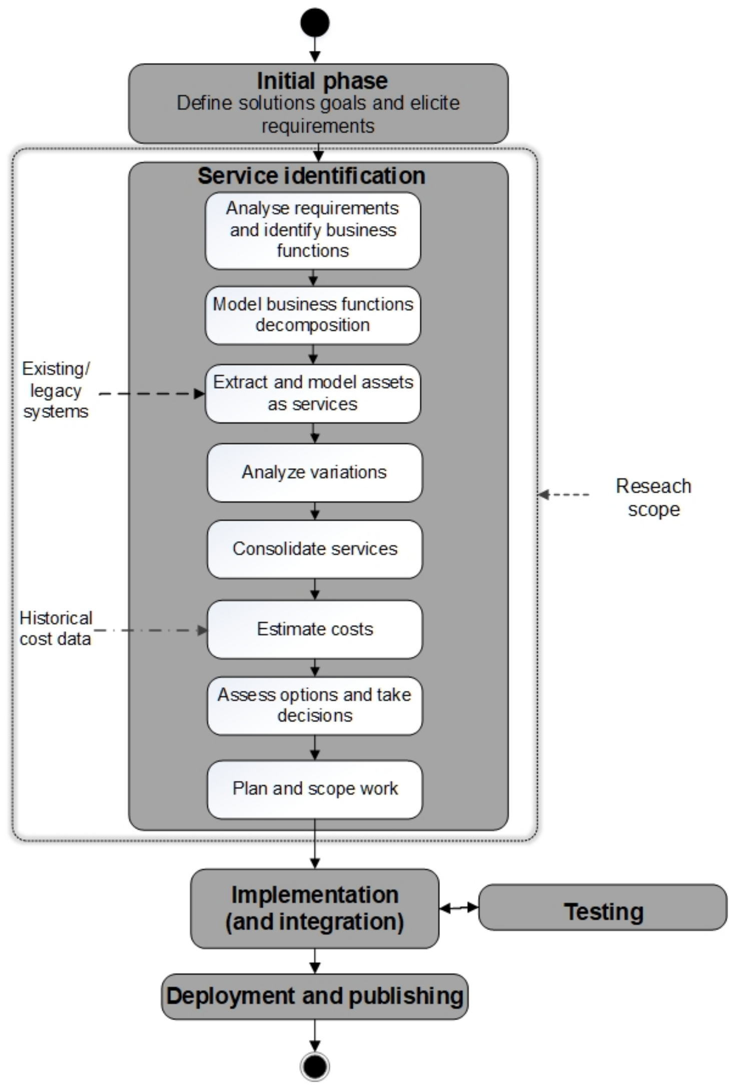

Initial phase
Define solutions goals and elicite requirements
Service identification
Analyse requirements and identify business functions
Model business functions decomposition
Extract and model assets as services
Analyze variations
Consolidate services
Estimate costs
Assess options and take decisions
Plan and scope work
Existing/ legacy systems
Historical cost data
Reseach scope
Implementation (and integration)
Testing
Deployment and publishing

Figure 4.4: Service identification and modelling decision pipeline

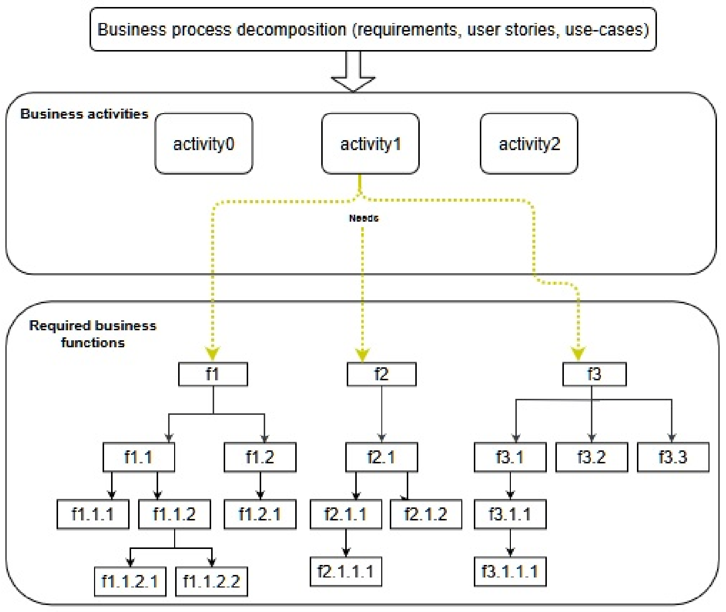


Figure 4.5: Top-down: business process and functions decomposition and modelling

Extracting and modelling assets: Starting from legacy or already delivered systems, SOA architect or a developer should analyze code. The aim is to identify precious functions that service similar business needs as the required functionalities. The focus is on function units that are in similar function areas and are used to realize similar business needs and features as the solution being designed.

In case it is hard to fetch function units in the code that contributes in realizing existing business functionalities for customers and with no design documentation, reverse engineering techniques on code and data models can be combined with automatic and manual code dependency analysis to extract the component list, functions, and dependencies. The method and call hierarchy graphs of an existing business function shows the natural decomposition of it from the underlying units. This analysis also allows filtering out infrastructure and utility assets in existing code and show whether there are duplicated function units and variations in business functions serving similar business needs in current applications.

At this stage, extracted function units in existing code are fine-grained and are combined with other units to serve a business function in a certain function area. Therefore, their natural composition to realize a business function in current applications should be modelled with similar graphs as required functionalities. Modelling assets makes it easier to compare between them as a first step, and to compare them with modelled requirements to maximize the reusability of existing function units and enhance granularity decisions. This can also enhance exploring potential services from existing applications to expand the portfolio being identified and plan them in the pipeline for potential customers.

Variation analysis: First, the required business functions in certain subdomains may require similar functionalities to realize them and certain variations which can be identified. This can be due to that a similar feature or function is needed for different types of entities or standards. As an example, applicant nationality, applicant type, or loan product type for loan application functionalities.

On the other side, existing assets that are modelled can also have such case due to the incremental development and targeting certain types of customers. They should be compared together and to required functionalities to spot shared functionalities and deviation levels (See Fig 4.6). Similarities and deviation levels can indicate a proper level of granularity in which existing assets can be used to implement a functionality and reduce development costs. This comparison between already developed business functions and required functions help choosing different boundaries between services on requirements graphs and compare between options based on cost KPIs and size of scope of services later in the decision pipeline.

For next iterations of applying the approach in design loops, business functions classification can help to first search the catalog for modelled assets in specific function areas to include them in the variation and similarities analysis and potentially reduce costs by deconstructing and constructing assembled services based on requirements.

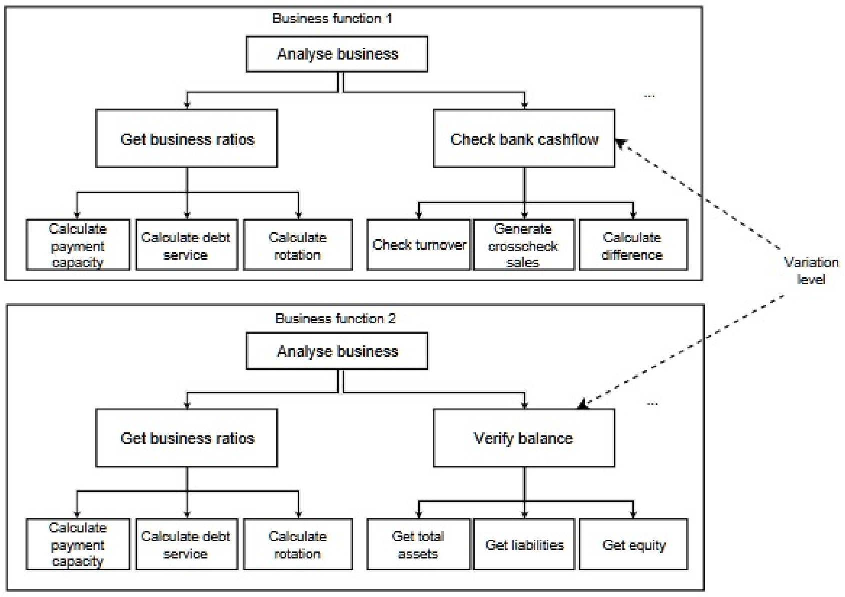


Figure 4.6: Variation analysis

Services consolidation: Comparing the required scope of functionality with modelled scope of functionality from assets includes checking whether there is a simple matching of required functionality and existing function or the variations to identify boundaries. This comparison involves the following cases:

- Business function can be realized by an existing function/service (simple matching)
- Business function cannot be realized by an existing function/service (no matching)
- Business functions similarities were found and the function can be partially realized by functions/services
- Business function can be realized by composing various functions/services
- Models can be reorganized based on variation analysis of assets to maximize reuse of existing assets and implement required functionality. This may involve the need to refactor existing services and functions
- Some functionalities in assets graphs can be exposed as services to expand the portfolio. This may be the case when it is reused multiple times in existing systems to realize similar or various functionalities or when migrating legacy systems to establish a portfolio for future solutions

The knowledge-base, namely rules by observations can be queried to support architects with a recommended action and a reference if needed. For example, if a service is found, the observation may concern performance, mainly due to current consumption or the data size. Therefore, a reuse may not be suitable. Architects can query rules to fetch recommendations and risks.

Moreover, existing implementation of units may not be fully suitable for the later implementation of service operations, or may need to be refactored first to realize different functionalities or to enhance coupling. This can also be the case for an outdated version or language or due to planned patching.

Iterative execution of variation analysis and service consolidation may be required to look up services per function area or business process.

In the end, the levels of granularity can be defined on the (expanded) requirement graphs to show where candidate services are and estimate costs, so that final decisions can be taken (See Fig 4.7). When having different options, each of them will include a set of candidate services and their levels of granularity on graphs.

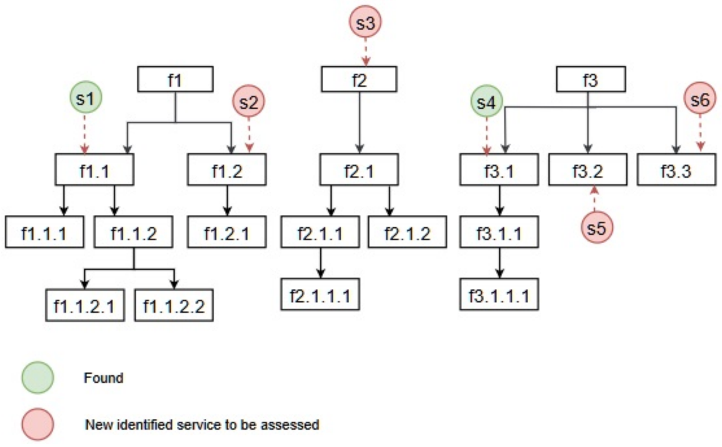


Figure 4.7: Service consolidation

4.2.2 Granularity and cost estimation metrics

To implement the decision support model for the identified services, we need to have proper granularity and cost estimation metrics. In this section, we propose suitable metrics to be used in the decision pipeline, namely, to make cost-effective decisions and plan services as part of the portfolio management lifecycle.

First, to indicate how granular a service is with respect to the portfolio or the set of required functionalities, we need to quantify the scope of functionality it covers.

Considering the following:

P is the set of the required high-level business functions (Portfolio)

S is the set of candidate services identified on the graphs to realize P

F_P is the set of all function units in the graphs of P

F_p is the set of all function units from a high-level function p in the set P

F_s is the set of the directly realized functions of a service s (level of granularity)

F_f is the set of following functions to a function f

To define granularity and cost metrics, we use the example in Fig 4.7, which shows the set of high-level business functionalities and identified levels of granularity on graphs.

4.2.2.1 Service size and granularity size-based metric

SLOCs, KSLOCs, Function Points (FP), and Story Points (SP) are often used to size functionalities or required efforts to implement it in Agile projects. The choice of the metric to size function units is left to the software company. For simplicity, we will use SP in our proposed formulas. However, this can be replaced by any other preferred measure.

A service can cover one or many functionalities, and each of those may need various function unit to be realized following the decomposition routes. Therefore, we need to define the size of a service. On functional scope graphs, a unit f can be either a leaf or a node with outgoing arrows.

Considering SP(f) is the size of f. The size of a leaf unit f is SP(f) and it doesn't need any composition efforts as it has no following units. While for a node unit f, the size of f is a sum of sizes of all following units and the size of composition such as invocation and state controls for all following units. Since we have function decomposition graphs for modelling the required set of business functions, we define the size of a unit f in the graph as a recursive formula with a base-case f as a leaf.

Function size metric:

$$Size(f) = \begin{cases} SP(f), & \text{if } f \text{ is a leaf unit} \\ \sum\limits_{\mathrm{f}_i \in \mathrm{F}_f} Size(\mathrm{f}_i) + \mathrm{CompSize}(f_i), & \text{otherwise} \end{cases}$$

Where F_f is the set of all following functions of f, and CompSize is the given SP to perform the composition for f. This sizing is important to estimate the effort required for implementing the logic that is not directly realized by a service in P.

Service size metric:

$$Size(s) = \sum_{i=1}^{|F_s|} Size(f_i)$$

High-level business function size metric:

$$Size(p) = \sum_{i=1}^{|F_p|} Size(f_i)$$

Service portfolio size metric:

$$Size(P) = \sum_{i=1}^{|F_P|} Size(f_i)$$

To quantify the granularity, we can estimate the size of a service s in relation to the size of the required scope of functionality to realize the high-level business function p in the set P, Zhan et al [34] functional size metric can be generalized as follows:

Service granularity-size metric:

$$SM(s) = 1 - Size(s)/Size(p)$$

Where p is the high-level business function preceding s.

For the example in Fig 4.7, the granularity of s3 is 0 which is the maximum value indicating that the service cannot be more coarse-grained in respect to f2. This sizing can also be done in respect to the whole set of functionalities P.

The granularity-size metric indicates how big the functionality a service implements in respect to the required business function.

If a service s is part of multiple graphs and therefore can contribute in realizing more than one high-level business function p in P (two or more incoming arrows to a node in F_s), then this size can be calculated for each of those paths to indicate the relative size to each business function.

4.2.2.2 Service granularity-scope metric

Service granularity is often looked at from a functional point of view. The functional scope included in a service can be indicated by how many functionalities can be directly or indirectly realized by the service. Zhan et al also propose scope-based metrics for the functional components in relation to the system as a whole. Those are the width and depth of the component to the overall high-level feature. However, they concluded that each of his metrics alone doesn't give a good quantification for the component functional scope or the amount of functionalities realized by the component with respect to the high-level feature. Therefore, he proposed a metric to combine both of them. We can generalize their optimized metric to apply it in the context of services rather than the functional components as follows:

Service granularity-scope metric:

$$SCM(s) = 1 - (n_i - 1)/max(1, n_p - 1))$$

Where n_i is the number of all following units to a service s (following units of functions in F_s), and n_p is the total number of units from p crossing s until all covered leaves by s

The value 1 indicates a maximum fine-grained service and 0 indicates a maximum coarse-grained service. In our example in Fig 4.7, s3 has the value 0 in respect to f2, because it directly implements the high-level functionality f2. However, s5 and s6 have the value 1 indicating maximum fine-grained services. The metric can also be calculated in respect to the total number of units in P instead.
The scope metric indicates how many unique functionalities a service s can realize in respect to the total number of functions to realize a business function.

Now, we established the granularity indicators, in the following section we propose an optional cost estimation model to be used to estimate the various costs to create services in a portfolio P.

4.2.2.3 Cost drivers

According to COCOMO II [30], code size has an exponential effect on effort and size due to increasing complexity. An important aspect of service granularity that influences costs is the service complexity, namely cohesion complexity, due to the size of functionality implemented by a service. We propose to use an exponent *exp* in the range of calibrated values [1.05..1.20] by COCOMO II[1]. The bigger the size or scope of service, the higher will be the cost to implement it which will be reflected by using *exp* as an exponent to the size.

[1]Following *https://en.wikipedia.org/wiki/COCOMO*

Furthermore, junior developers may spend more time on implementing and combining services, but their hourly rate can be less than senior developers. Experience can be reflected by introducing two or more separate cost rates or by multiplying the efforts by an extra Estimated Adjustment Factor (EAF), which usually has a nominal value of 1.0 and can be increased to take the case of junior developers into consideration.

To extend COCOMO II, which is a component-based cost estimation method and consider the composition in SOA to deliver a business solution to customers, we suggest using block-based estimation formulas. It is important to note that the value of the intermediate cost parameters can be a *monetary unit*, *man-hour*, or any effort unit preferred by the software company. The resulting cost values will be in the same unit. Moreover, sizing functions, services, or composition logic should all be in one unit, e.g. SP.

4.2.2.4 Implementation cost

For a service s defined on the functional graph(s), the cost of implementing this service depends on the directly realized functionality, i.e. the function unit at which the granularity is set. Also, whether the size is measured by SP, SLOC, or KSLOC, a rate *iRate >0* is proposed to represent the rate unit. For example, when using KSLOC, the rate is the effort required to implement 1 KSLOC. We also add extra costs *iConst >0* to consider service interface, description, and publishing. This can also reflect unit testing for the functionality or service deployment costs.

Service implementation cost metric:

$$ICS(s) = iRate * Size(s)^{exp} + iconst$$

This calculation should exclude the size of SOA services that can be reused or the implementation logic size of existing code and only considers the constant value. The refactoring effort for existing code is discussed later.

The total implementation cost of the candidate service list to realize P is proposed to be the sum of implementation cost for all services in S, then:

Total implementation cost metric:

$$IC(P) = \sum_{i=1}^{|S|} ICS(s_i)$$

4.2.2.5 Service composition cost

In case of some services need to be combined to realize a functionality (e.g. s1 and s2), the composition effort should be calculated. Already existing composite services that can be reused should be excluded at a later time of adopting the approach and having service assets. The more the number of services the more the composition effort to be spent due to the level of complexity. Therefore, we can use the exponent to reflect the increase of complexity as the number of services increase.

A constant cost *scConst* ≥ 0 may need to be paid, e.g. EBS, BPM engine, if this is yet to be paid for the solution, and the composition rate scRate >0 are used to estimate the composition cost for the minimal composition size value. This rate is different from the implementation rate, as different tools, programming language, or people that are involved in this task.

Composition cost metric for a business function p:

$$CCp(p) = \sum_{f \in \mathrm{F}_p} scRate * CompSize(f)^{exp} + scConst$$

Where F_p is the set of the functions that are not realized by a service for the high-level function p.

Total composition cost metric:

$$CC(P) = \sum_{\mathrm{p} \in \mathrm{P}} CCp(p)$$

Other development efforts on the activity level can also be added,if needed, such as the composition of high-level functionalities and invocation. This is the case when new SOA solutions are being designed and when there are already available SOA services.

4.2.2.6 Refactoring cost

In case an existing asset needs to be refactored before it can be reused, this effort should be estimated. This effort can increase when the size of the functionality to be refactored increases. We propose to use historical effort from functionalities of same size and the same exponent exp to reflect the increased complexity and efforts on size increase.

Considering F_r is the set of all following units to f_s that need a refactoring, and rc_f is the refactoring size for a function f in F_r, the refactoring cost for f_s that implements s:

$$RCS(s) = \sum_{\mathrm{f}_r\ \in \mathrm{F}_r} iRate * rc_f^{exp}$$

Total refactoring cost metric:

$$RC(P) = \sum_{i=1}^{|S|} RCS(s_i)$$

4.2.2.7 Total service development cost

The estimated total cost of the scope of functionality in P can be calculated by summing values for implementation, composition, and refactoring costs as follows:

$$TC(P) = IC(P) + CC(P) + RC(P)$$

The resulted estimate doesn't consider any extra risks, such as people leaving teams or other cost-sensitive factors. Those can be reflected by extra EAFs if needed.

4.2.2.8 Minimum maintenance cost

SOA encourages the implementation of a shared functionality by various applications and business services in a separate IT service or an operation in service. However, this may not be ideal, especially if the size of functionality is small and other aspects such as the performance, shared data models, and cohesion complexity are considered. It can also due to setting improper levels of granularity and missing chances to reuse operations in existing services when developing new applications. When technology or programming language change for underlying SOA services, there will be higher costs in case of redundancies and overlapping functionalities between services. The bigger the duplication size, the higher is the cost due to the complexity. Therefore, we consider using the exponent and the rate mRate >0.

Total maintenance cost metric:

$$MC(P) = \sum_{\mathrm{f} \in \mathrm{F}_P} max(0, n_f - 1) * mRate * Size(f)^{exp}$$

Where n_f is the number of implementations for a function f in P.

4.2.3 Service portfolio decision model

Use of past efforts in local cost estimation: To take final cost-effective decisions for the modelled services on graphs, estimation to the cost for implementing, composing, and refactoring those services should be done. A comparison between the various options from an economic perspective should be done. The integrated granularity and cost indicators should be used to do this task.

The service size metric can help in sizing service based on the covered functional scope or functionalities and estimate the implementation cost. It is important when adopting the proposed implementation to follow a standard procedure for logging the efforts related to developing services, so they can be used in estimation and to organize resources and prioritize work for new services. This is also important as part of the service portfolio management, as this data can be used in analyzing the long term impact of creating services (granularity choices) on costs, compared to the original investment and benefits. However, time logs, code commits, and any backlogs can be used to build an overview of spent efforts in earlier created services of *similar purposes and function area*, or of *similar granularity and function area*, depending on the availability of already delivered assets. In order to size the covered functionalities in a service, historical efforts on available services and functionalities in sizing functionalities and therefore the service.

Moreover, the proposed granularity metrics can help in splitting and scoping work in the service pipeline and allocate resources. The granularity indicator can also show how much of the whole required functional scope a service covers, so the level of granularity considering the whole scope and when having different options can be compared to take final decisions.

Assessing and prioritizing needed services and functionalities: The resulting options from previous steps should be assessed, based on the local costs and granularity indicators by a dedicated team lead or portfolio staff to come up with the final candidate list of services and the modeled scope of functionality, namely, directed graphs. Special cases such as leveraging legacy can also be assessed whether this legacy code is suitable to be wrapped and exposed as services. This is also the case when there is a need to refactor an existing service or functionality. In this case, there are multiple options and costs. The refactoring cost in one option should be compared to implementation cost in another (See Fig 4.8). If a service is to be implemented instead, the initial level of granularity might change to discard existing service. At the end, spent costs can be monitored against the initial estimated investment once Agile teams started to deliver those functionalities.

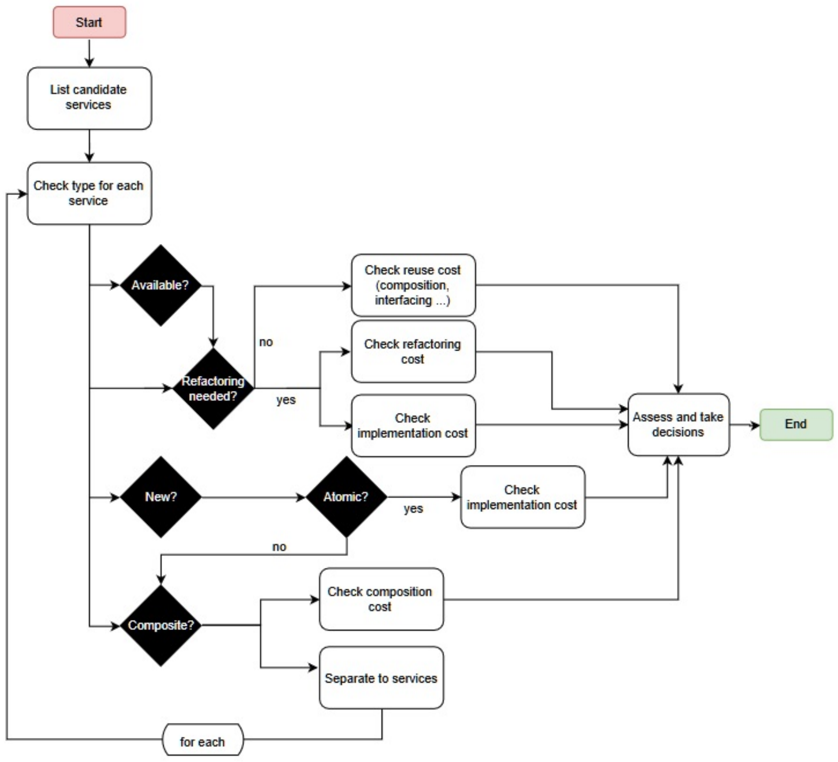


Figure 4.8: Cost estimation and decision making

After assessing work in the decision pipeline, it can be planned and published with the models and services description in the service pipeline, so other development teams can access them. This task is often done by a portfolio manager, who can see models and their linked efforts and the size of identified services to plan for investment and dedicate resources.

4.2.4 The continuous learning

Adopting the proposed modelling and decision making approach for every development iteration and for the various products allows to establish and maintain an up-to-date knowledge-base, which contains the models of already delivered business services and functionalities in previous releases, past spent efforts, and references. The supporting knowledge can be fetched when designing new solutions to be compared with required business functions and take reliable and cost-effective granularity decisions and evolve both solutions and service assets to align with business and optimize costs.

As a result, having a continuously updating knowledge-base from combining Agile approach with the proposed solution, makes the solution work as a self-learning feedback tool which helps development teams to learn from past support and past design decisions when they take new decisions and estimate costs, as well as for service portfolio management. To get feedback from the portfolio management cycle to the development cycle in able to enhance decisions and mitigate future risks, raised issues and customer feedback as well as the resulting increased costs and efforts from the monitoring process in the portfolio management cycle should be reported by portfolio management staff. There must be a standard process in place and dedicated participants to receive this feedback and maintain the rules in the knowledge-base.

4.3 The rationale behind proposed implementation

Applying a hybrid service identification, classification, and modelling activities facilitate leveraging and reusing existing assets and composite services that can contribute in delivering new solution. This way local costs are optimized through reusability and future maintenance costs are reduced when solutions and service assets evolve. In Agile, requirements are processed iteratively and are broken down and scoped in backlogs to various teams. As a result, the available requirements already presume architectural assets and implementation through that specific scope of knowledge. Therefore, different development teams may model requirements and services in a way that doesn't align with the capabilities of existing assets and affect architecture evolvability and assets sustainability negatively. Even similar architects and developers may lose the context between the time design decisions were taken in previous iterations and current iteration.

Furthermore, reliable and up-to-date feedback enhances the cost estimation when designing new solutions. Therefore, integrating proper granularity and cost estimation metrics is important to take cost-effective decisions as well as to learn from past decisions and their impact on cost before implementing new features and functionalities.

Use of existing systems and already delivered business services: In the context of legacy and existing systems, extracting the natural composition of function units to realize business functions in existing code and combining them with requirements in the design loop can support creating new solutions from existing assets and reduce the cost of establishing the portfolio of services from envisioned requirements.

On the other hand, without having an evolutionary overview of assets, modelling services with a top-down process decomposition is not enough to maximize the reusability of existing code and services and to take optimal granularity decisions for future. Therefore, leveraging already delivered business services and asset and the variation analysis are essential for managing software lifecycle and optimize costs. Also, teams can learn from each other and can avoid modularity choices that caused pain in the past. Arranging teams around products can help them enhance decisions when performing the variation and consolidation checks.

Modelling technique: For implementing the approach, we used a directed graph to model the composite scope of functionality for a business service and functional requirements. This graph can show a natural composition of business functionalities and allows for shared functionalities to be modelled unlike feature trees proposed by Zhan et al [34]. It can be used next to functional documentation and as an extra knowledge in the service catalog to show how functions are assembled from IT services. Visualizing functionalities support variation analysis and service modelling. Therefore, it speeds up leveraging existing assets that were used in the past to assemble similar business functions and when migrating different legacy systems to building blocks (SOA services). Furthermore, creating similar graphs at an abstracting level allow business and marketing people to understand them and can help development teams later in testing and interpreting customer feedback. They can also include external services and COTS in a similar way. For companies starting with SOA and having legacy code, IDEs nowadays often offer methods and call hierarchy overview and diagrams for existing methods.

Classification: The business-domain-driven classification for both the functional scope and services is inspired by SOMA way of identifying services in one SOA solution [3]. In our project, it offers a better cohesion and proper abstraction that can enhance the coordination between the various IT stakeholders working on different projects and speed up leveraging suitable assets. It also helps in organizing small development teams around business domains or function areas, and assign dedicated people for monitoring and managing a set of solutions and service portfolios. Finally, it eases the alignment and reuse of IT services to create business services and solutions to customers in distinct markets and business lines.

Continuous learning and knowledge management: Adopting our proposed approach with Agile development adds the continuous learning and knowledge management dimension. This additional dimension, compared to existing service identification and modelling methods, connects Agile development activities and modelling with service portfolio management through the knowledge management cycle and the continuous learning. The shared knowledge and feedback between those two when solutions evolve allows companies to sustain their solutions and service assets while having self-organized and independent development teams. With existing approaches, this connection doesn't exist, which doesn't allow for both service portfolio management and development processes deliver feedback to each other effectively, to manage the evolution of service portfolios and mitigate future risks.

Observations and decision rules: Over time, decision rules can enhance the learning and maturity of decisions in design loops and work as a lightweight decision logger and connect the design pipeline with portfolio management through the analysis of issues and spent costs. The rationale behind proposing rules and adding recommended actions and references is that required granularity decisions not specific to one application in software companies [24, 37]. Also, in one application and with Agile incremental development, they recur and they can be enhanced by analyzing past pain points to derive risks such as increasing cycle time and development efforts, instead of increasing future solution re-engineering costs. Moreover, observations in the context of service reuse in consideration with the reuse plan of a created service can only be derived when this service is used in solutions while indicators such as the number of customers using the service give undesired values.

Even in short customer feedback loops, when working on complex software products, losing context between the time when decisions were taken in previous loops and new loops is common [6, 23]. Attempts to have a shared context between stakeholders often focus on how to solve problems, rather than why it was done and how to enhance it. Also, many of us join development teams and get access to loads of documents & descriptions of components in the system and huge architecture diagrams that are hard to revisit and to update periodically. Also, those don't give a real explanation of what drove the decisions and actions and how solutions evolved to avoid repeating mistakes.

4.4 Long term KPIs and impact of granularity decisions on business

In the proposed approach, we integrate development cost metrics and monitor measured costs for delivered and new solutions. However, our decision framework allows to quickly integrate long term KPIs which usually aim to measure the impact of short-term and already taken design decisions on the quality of services offered to customers and business over time. Those indicators offer extra feedback by undesired values to derive business and cost-related decision rules and maintain additional knowledge about developed solutions and services. This additional knowledge offers an extra enhancement to the learning process and design decisions in our framework.

Long term KPIs usually aim to encourage a win-win situation, in which benefits to both software provider and customers are delivered. For instance, when investing in creating and expanding services in a certain business domain or function area, the number of services reused in this domain and the customer satisfaction through employing IT assets to deliver the desired business functionality can give a good indication. Other KPIs can measure the number of business disruptions due to (IT) service incidents, service downtime, or incident resolution time to check whether they comply to SLAs.

The service provider can decide which of those global measures are critical to business and assign owners for the measurement and reporting activities. Automating the calculation of critical indicators can guarantee a real-time impact analysis result and efficient reporting to relevant participants. However, the key is to not overdo it and to not prevent teams from working effectively and independently, delay deliveries, and restrict innovation.

Since business functionalities are assembled from various SOA services of different technology, improper boundaries between services can lead to a long service disruption for customers when one SOA service need to be replaced due to requirements and technology change. It becomes also hard to assign a KPI for certain services. Longer development cycles and time to introduce changes when software evolves can also be good indicators for having bad design decisions.

In Table 4.1, we map common business and IT drivers for using SOA to deliver business value to customers and measurable global KPIs that are relevant to service monitoring and management. Those are derived from best practices in literature [12, 8, 28, 31] and are reviewed by Mr. I. Hooks partner and director, and Mr. F. t. Beek quality and ISO manager at Stachanov.

Category	Business drivers	IT drivers	KPI
Financial perspective	Development cost reduction and revenue increase	Maximum reuse of current IT assets. Assets should be designed to be flexible in use. Common information and messaging models should be defined to ease accessibility and discoverability	1. Percentage of customers using service 2. Percentage of services reused 3. Cost of new process rollout 4. Return On Investment (ROI) 5. Maintenance cost
Strategic and internal processes perspective	Surviving competitors pressure and higher business flexibility	Faster response to business demands (business agility), spending more time on innovation rather than on maintenance, lower time-to-market, and competitive prices through flexibility in change and use	1. Percentage of customers using service 2. Percentage of services reused 3. Percentage of new customers over a period of time 4. Resolution time meeting SLA & breach rate 5. Time-to-market 6. Maintenance and re-engineering cost 7. Percentage of innovative ideas and experiments in new markets
Customer and service level perspective	Service level improvement	Improved service delivery, faster response to change and flexibility in change, and solution sustainability	1. Speed of leveraging information over current assets 2. Resolution time meeting SLAs & breach rate 3. Application and service up-time: number of business disruptions due to (IT) service incidents 4. Ability to replace services and update applications (overnight) with minimum interruption to customers 5. Number of accepted complaints about service 6. Customer satisfaction/feedback (e.g. NPS) 7. Percentage of lost customers
Partnership and collaboration perspective	Seamless integration with partners	Reduction in integration time and overhead, and the ability in leveraging shared IT assets to create high-value assets	1. Number or percentage of business services delivered through integrated internal and external services 2. Speed of leveraging information over current assets 3. Time/cost to integrate external service
Knowledge management and IT-Business alignment perspectives	Information overload reduction and supporting knowledge availability	Separation of concern through information distribution, offering right knowledge to IT people at the right moment to make decisions, classifying and modeling IT assets to enhance reusability and reduce mistakes and indirect duplication of service capabilities	1. Speed of leveraging information over current assets 2. Percentage of instances and overlapping services 3. Percentage of services to be refactored or phased out 4. Percentage of SOA solutions to be re-engineered upon change 5. Average time until a problem is identified 6. Customer satisfaction/feedback (e.g. NPS)

Table 4.1: Key Performance Indicators for service portfolio management

Chapter 5

Case study

In this chapter, we first summarize the preliminary qualitative research at the host organization to analyze the problem and gather enough information to apply the approach. Next, we demonstrate how our proposed granularity decision making approach with a use-case at Stachanov.

5.1 Preliminary research at the host organization

The aim of those activities is to analyze the problem and collect enough arguments and evidences that would either endorse or falsify the need for adopting SOA and having a supporting approach for managing the evolution of service portfolios, as well as to collect initial requirements.

5.1.1 Interviews and group discussions

Interviews in a one-to-one setup and group discussions among the involved stakeholders were conducted at Stachanov. This helped us gain an understanding of the application domain and problems and to collect the important business objectives and stakeholders point of view and arguments.

Participants in interviews and group discussions were: software architect & developers, management board members CEO/COO/CFO, and members in the operation team (See Table 5.1).

Role	#
Software architect	2
Senior developer	5
Junior developer	2
Management board	2
Operation staff	2

Table 5.1: Interviewees

Open-ended questions in the one-to-one interviews, with follow up questions were used. Main questions were to seek situations in which developers had to solve a problem they "believe" is due to the current architecture and inflexibility of change, or had to re-implement a functionality for a different customer rather than choosing to include it at the platform level. We also used follow up questions to see how they believe such case can be avoided and whether their choices were driven by differences in requirements and incremental receiving and processing of requirements, having a monolith architecture and bounded time per development cycle. This was important, as some of those problems may also occur due to developers discipline, tight-deadlines, lack of experience, and insufficient testing.

Moreover, focus groups were selected to hold group discussions which mainly focused on having the management and business analysts with IT stakeholders in the discussion. The aim of those discussions was to discuss the problems in depth in a language that is understandable by the management and see how expressed problems by the various IT and business stakeholders in first interviews are holding the business back. Such discussions revealed how organizational learning and knowledge transfer are taking place at the company, the environment, and the culture.

After those interviews and discussions took place, classification of business objectives and issues were identified and crosschecked to come up with the main business objectives and Stakeholders context at the company. Furthermore, we identified the main challenges for adopting SOA and creating a set of services from current applications that can be base for future customers. The results of the analysis are presented in the next section.

5.1.2 Results

Stakeholders context: In Table 5.2, we present the identified stakeholders in the studied case and concerns

Furthermore, generally, there is good coordination, issues and tasks logging, and short daily meetings between IT and business people to assign tasks and coordinate work.

Business dimensions: As a result of the preliminary discussions, a set of business objectives were identified as follows:

- Increase reusability over re-implementation and growing size of software assets and applications
- Maintain sustainable assets
- Reduce time-to-market
- Reduce (SLAs) resolution time/meeting
- Gradual adoption of SOA and creation of services
- Taking advantage of already created applications in creating services and not wasting already invested money

We also identified a set of initial business goals the management would like to achieve iteratively:

- Having pre-defined risk assessment and credit scoring services for helping banks mitigating risks of granting loans and subsidies and to include them in bank simulation and training solutions as well
- Having generic financial model and calculations with standard functionalities
- Having generic supporting features from the platform as separate business services. This way they can be patched and replaced separately due to the use of open-source plugins that needs regular patching

Quality of services and portfolio management: Stachanov teams do a good job in logging work and issues and classify them in their available jira system. They also have few processes for monitoring service quality, managing SLAs, and reporting quarterly to some of their customers on application availability, the number of incidents and SLA response times to incidents and change requests [1]. They hold daily morning meetings to discuss responsibilities and prioritize work. However, there is not many efforts spent in documenting design decisions and a shared knowledge-base between IT and business teams about existing solutions in banking and micro-finance sectors.

Challenges in establishing a service portfolio from legacy systems: There are various challenges in identifying services from current applications and for SOA adoption. Those challenges are as follows:

- Lack of design & technical documentation for existing applications
- Lack of shared knowledge-base and categories for software assets and business functions to be accessed when assembling new business applications
- Lack of modularity and interfaces in existing applications: monolith architecture
- Knowledge about the capabilities of current assets is owned by experienced developers and architects. There is no clear overview of what business functionalities are valid to assemble new solutions
- Many business functionalities are re-implemented in different code branches and applications as part of application business logic according to developers
- The platform itself is forked and evolved with certain applications: sometimes merging back changes to the trunk in subversion is delayed or never done. This is due to the short-term fixes rather than maintaining suitable modularity for functionalities and setting dependencies on abstraction rather than on evolvable code

Furthermore, existing resources are limited and should be used efficiently. Therefore, it is important that the approach can support them in organizing IT and business people in an efficient way.

[1] *https://iso.stachanov.com*

Stakeholder	**Details**
Business owner (CEO, COO & partners) and business analysts	1. The targeted market has certain standard business demands that can be separated from banking variable lending policies and plans 2. Management usually takes the initiative and approach their customers with new features and services in credit risk assessment and credit scoring domains. Mainly, because banks are in need of such services and they don't have explicit requirements and an need expert opinion and expert solutions 3. Business agility becomes an issue as products evolve and more requirements are delivered 4. The platform capabilities are already consumed to the max. Recently, there are no standard business functionalities added that can be reused to cut on development costs in new solutions, while the impression is that many would have been done by developers to extend the set of services the platform can offer 5. After few years of implementing web applications in the targeted market, necessary patches and upgrades to critical components in applications for Java versions and backend platform are delayed. This is due to the resulting size of current applications, whilst those applications have convergent characteristics and features 6. The company is starting up a new branch in Africa to deliver loan origination BPM solutions and risk assessment services to micro-credit banks there. It would be beneficial to have pre-defined blocks to offer solutions to such recently growing market. However, it is yet unclear how to reach there and whether they can benefit from already developed applications and learned lessons from current situation
Software architects and developers	1. As requirements change and applications grow, it becomes hard to fix bugs and introduce change. Over time, refactoring and replacing plugins with their integration code in one application to reuse certain business functionalities for different features costs a lot of time due to the size of code. It also needs careful testing as those applications are already in production and components responsibilities are wide 2. Various developers work on projects of similar types with some developers come and leave 3. No shared & standard knowledge-base exists for developers and also between IT and business people 4. New team members find it difficult to understand which features are recurring and how to separate them so they can be reused. Moreover, the big responsibilities of functionalities require teams, especially junior ones to have a big context in their mind while working on a task and introducing changes with no proper modelling for assets 5. No clear feature and component ownership exists, especially for generic features that are coarse-grained and can be separated in the platform instead 6. They are aware of that some industry-standard features exist and that some of them are re-implemented or included as part of custom features, due to the changes in requirements and the various developers and interns working on separate projects 7. According to senior developers and main platform architect, some of the existing functionalities can be decomposed into custom and agnostic functionalities. The later can be abstracted and encapsulated to reduce dependencies and enhance the reuse and changeability. However, when developing a new solution, it costs less time to re-implement rather than analyzing existing code written by others
Operation team	1. The team doesn't contribute much in deploying applications. However, members are responsible for hosting and patching operating systems and tools, as well as setting up monitoring tools for performance and availability and creating reports following the implemented ISO standard (9001, 14001)
Customers	1. Customers are not aware of the architecture and the size and modularity of software components in their solutions. They care about the agreed business services, which should meet their expectations and respond to their demands, with the competitions out there that can offer such solutions 2. As software evolves, SLA response times for changes and fixes increase. Adding features for extra loan product becomes slower, according to the available quarterly reports and interviews with management

Table 5.2: Stakeholder context

5.2 Applying the approach

To demonstrate the approach, we selected two loan and financial services applications at Stachanov. Those were used mainly for the bottom-up activities to extract and model precious assets that can fulfill management requirements in having a standard portfolio of financial and credit risk services, and identify generic services from applications and the platform.

5.2.1 Business goals

The business goal is to offer credit risk and financial analysis services to banks and financial institutions through multiple channels, including business process management solutions, banking simulation, training consumers, and other online service consumers to manage credit risks and orient clients to different loan products.

5.2.2 Service identification and modelling decision making

5.2.2.1 Top-down: requirement analysis

Domain decomposition: First, we decomposed the loan underwriting domain based on various high-level flows of activities for current customers and management view. To demonstrate the approach we will take an example from "Credit risk assessment" subdomain (function area). The complete list of the identified functional area can be found in Appendix B.

Process decomposition: Second, for credit risk assessment process of a credit applicant, the subprocess, namely high-level activity, "Manage business risk" includes many business functionalities that are offered to banks and financial institution to help them study the applicant business situation and mitigate risks of approving the loan. In this step, we consulted Mr. A. Koch[2] from Stachanov to understand all standard business functions that are needed to manage risks and have enough information to assess the credit application. Table 5.3 shows the subprocess "Manage business risk" and part of the required business functions to satisfy the assessment.

Small Enterprise Applicant (SEA)
Manage business risk
1 analyze balance details
1.1 process balance assets
1.2 process liabilities
1.3 calculate equity
2 analyse business
2.1 manage profit and loss
2.1.1 calculate sales
2.1.2 calculate purchases
2.1.3 calculate expenses
3 score applicant with altman-z model
3.1 calculate business ratios
3.2 retrieve standard weights
3.3 calculate weighted score
4 evaluate guarantor
4.1 analyze physical collateral
4.2 analyze stock market value

Table 5.3: Decomposition of required business functions

5.2.2.2 Bottom-up: existing systems analysis and assets modelling

Since the company has no SOA services yet, there was a need to analyze the selected applications and the platform itself to extract and classify potential business functions from existing code, so they can be used to cut on costs and not waste the already invested money in the studied business domains. The cases are as follows:

A *small enterprise loan application Fin1*

[2]Mr. Koch has more than 22 years experience in offering credit and financial risk assessment services, and giving training course for bank agents: *https://www.linkedin.com/in/andré-j-m-koch-60315562/*

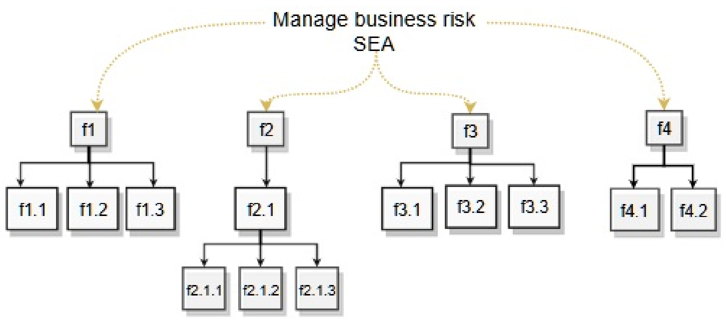


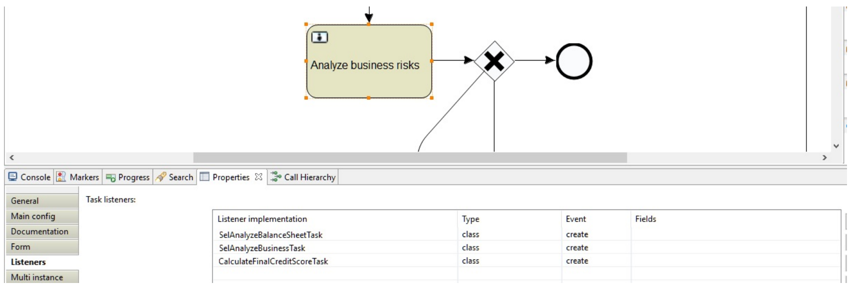
Figure 5.1: Modeled functional scope

B *small medium enterprise loan application DCB*

The aim is to extract the function units that implement a business function and model their current composition in the code, so they can be compared together and to the required business functions, before identifying serviced and estimating cost.

Reverse engineering and code analysis: To be able to extract precious business functions from existing function units in delivered applications, we applied reverse engineering and code analysis activities using MySQL workbench, INTELLIJ IDEA, and manual work, to both relational databases and source code, to understand the boundaries between high-level components and between existing features and to extract the list of Grails plugins that are also integrated. It revealed what utility and infrastructure components are part of the platform and how they are used in existing applications.

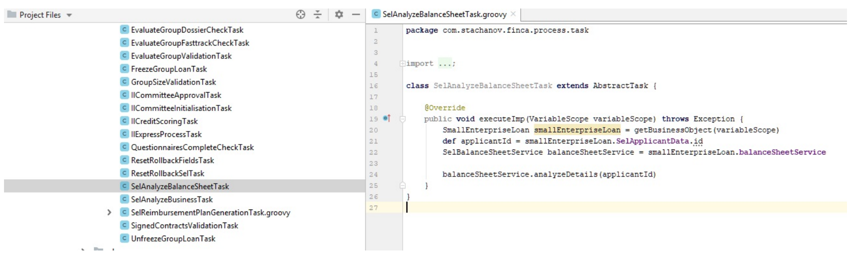
To discard existing business logic and data entities that are not part of the business activities, we collected existing BPMN files of the studied cases and we looked for similarities in workflow activity naming and grouped them. We focused on business and credit risk analysis and evaluation activities. To find the underlying functionalities of each activity, we looked into the implementations of Java tasks listener that are triggered by events on the high-level activities (See Fig 5.2 and 5.3). Those revealed the high-level business-functions in the call sequence, so invoked function units from the various existing Java and Groovy services can be traced back until all fine-grained basic units. This has revealed the natural composition to implement the identified business functions. To get an overview of the composition of each high-level function, the method hierarchy option was used to get the list of all invoked units until the fine-grained functions. While call hierarchy was used for each of the underlying units to spot whether they are invoked to realize different business functions.



Task listeners:

Listener implementation	Type	Event	Fields
SelAnalyzeBalanceSheetTask	class	create	
SelAnalyzeBusinessTask	class	create	
CalculateFinalCreditScoreTask	class	create	

Figure 5.2: Event listener implementation



EvaluateGroupDossierCheckTask
EvaluateGroupFasttrackCheckTask
EvaluateGroupValidationTask
FreezeGroupLoanTask
GroupSizeValidationTask
IICommitteeApprovalTask
IICommitteeInitialisationTask
IICreditScoringTask
IIExpressProcessTask
QuestionnairesCompleteCheckTask
ResetRollbackFieldsTask
ResetRollbackSelTask
SelAnalyzeBalanceSheetTask
SelAnalyzeBusinessTask
SelReimbursementPlanGenerationTask.groovy
SignedContractsValidationTask
UnfreezeGroupLoanTask

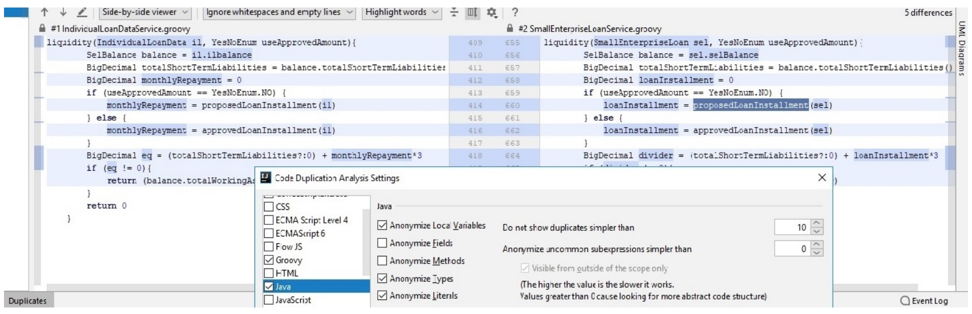
SelAnalyzeBalanceSheetTask.groovy

```
package com.stachanov.finca.process.task


import ...;

class SelAnalyzeBalanceSheetTask extends AbstractTask {

    @Override
    public void executeImp(VariableScope variableScope) throws Exception {
        SmallEnterpriseLoan smallEnterpriseLoan = getBusinessObject(variableScope)
        def applicantId = smallEnterpriseLoan.SelApplicantData.id
        SelBalanceSheetService balanceSheetService = smallEnterpriseLoan.balanceSheetService

        balanceSheetService.analyzeDetails(applicantId)
    }
}
```

Figure 5.3: High-level logic in the event listener implementation

All related Java and Groovy service and implementation files were grouped together to be compared in one Grails package. Clone detection type 1 & type 2 with anonymized literals and data types was used to spot redundancies in function units that are part of the natural decomposition of implemented business functions to find shared code. For example, the calculations of the financial risk ratios were found in various credit scoring, business analysis, and executive reporting functionalities.

On the other hand, we collected the domain classes that are used in similar functionalities and used UML diagram in INTELLIJ to spot the dependencies for each application. In Grails, each Groovy domain class ends up as a data table in the database. Therefore, we could add the data tables in MySQL workbench in a reverse engineering model for each of the cases. This helped us to use the comparison tool from MySQL workbench to see differences and analyze similarities in the model view. As a result



#1 IndividualLoanDataService.groovy

```
liquidity(IndividualLoanData il, YesNoEnum useApprovedAmount){
        SelBalance balance = il.ilbalance
        BigDecimal totalShortTermLiabilities = balance.totalShortTermLiabilities
        BigDecimal monthlyRepayment = 0
        if (useApprovedAmount == YesNoEnum.NO) {
            monthlyRepayment = proposedLoanInstallment(il)
        } else {
            monthlyRepayment = approvedLoanInstallment(il)
        }
        BigDecimal eq = (totalShortTermLiabilities?:0) + monthlyRepayment*3
        if (eq != 0){
            return (balance.totalWorkingAs
        }
        return 0
    }
```

#2 SmallEnterpriseLoanService.groovy

```
liquidity(SmallEnterpriseLoan sel, YesNoEnum useApprovedAmount){
        SelBalance balance = sel.selBalance
        BigDecimal totalShortTermLiabilities = balance.totalShortTermLiabilities()
        BigDecimal loanInstallment = 0
        if (useApprovedAmount == YesNoEnum.NO) {
            loanInstallment = proposedLoanInstallment(sel)
        } else {
            loanInstallment = approvedLoanInstallment(sel)
        }
        BigDecimal divider = (totalShortTermLiabilities?:0) + loanInstallment*3
```

Code Duplication Analysis Settings

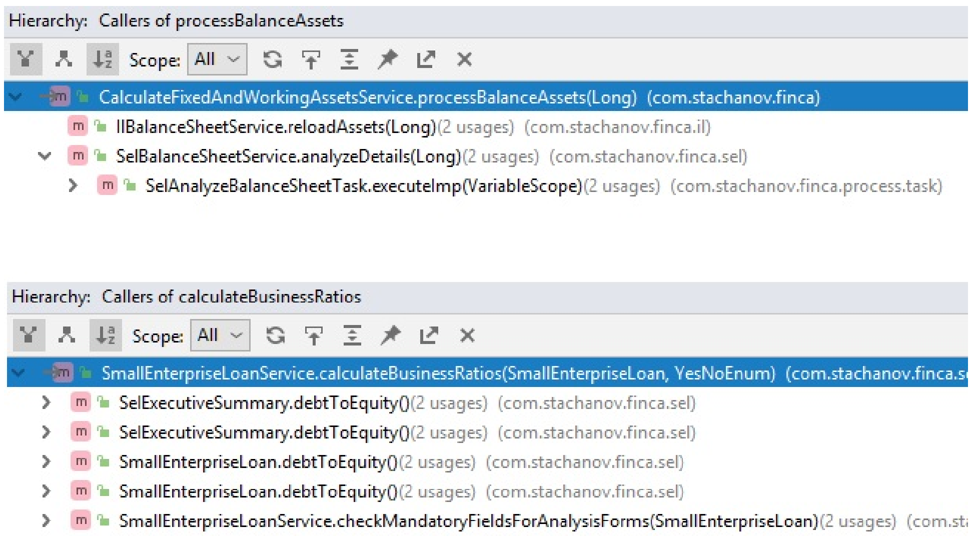
☐ CSS
☐ ECMA Script Level 4
☐ ECMAScript 6
☐ Flow JS
☑ Groovy
☐ HTML
☑ Java
☐ JavaScript

Java

☑ Anonymize Local Variables
☐ Anonymize Fields
☐ Anonymize Methods
☑ Anonymize Types
☑ Anonymize Literals

Do not show duplicates simpler than: 10
Anonymize uncommon subexpressions simpler than: 0
☑ Visible from outside of the scope only
(The higher the value is the slower it works.
Values greater than 0 cause looking for more abstract code structure)

Figure 5.4: Clone detection



Hierarchy: Callers of processBalanceAssets

```
CalculateFixedAndWorkingAssetsService.processBalanceAssets(Long) (com.stachanov.finca)
    IIBalanceSheetService.reloadAssets(Long)(2 usages) (com.stachanov.finca.il)
    SelBalanceSheetService.analyzeDetails(Long)(2 usages) (com.stachanov.finca.sel)
        SelAnalyzeBalanceSheetTask.executeImp(VariableScope)(2 usages) (com.stachanov.finca.process.task)
```

Hierarchy: Callers of calculateBusinessRatios

```
SmallEnterpriseLoanService.calculateBusinessRatios(SmallEnterpriseLoan, YesNoEnum) (com.stachanov.finca.s
    SelExecutiveSummary.debtToEquity()(2 usages) (com.stachanov.finca.sel)
    SelExecutiveSummary.debtToEquity()(2 usages) (com.stachanov.finca.sel)
    SmallEnterpriseLoan.debtToEquity()(2 usages) (com.stachanov.finca.sel)
    SmallEnterpriseLoan.debtToEquity()(2 usages) (com.stachanov.finca.sel)
    SmallEnterpriseLoanService.checkMandatoryFieldsForAnalysisForms(SmallEnterpriseLoan)(2 usages) (com.sta
```

Figure 5.5: Call hierarchy

of those activities, we ended up with the various composition and underlying function units of similar business functions in the two studied cases. Those are modeled using directed graphs to spot deviation levels and come up with a final list business functions and units to realize them (See Fig 5.6).

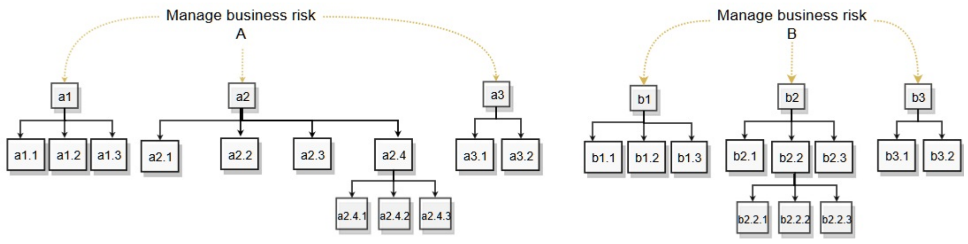


Figure 5.6: Modelled assets

A	**B**
Manage business risk	Manage business risk
1 **analyze balance details**	1 **analyze balance details**
1.1 **process balance assets**	1.1 **process balance assets**
1.2 **process liabilities**	1.2 **process liabilities**
1.3 **calculate equity**	1.3 **calculate equity**
2 analyse business	2 analyse business
2.1 **calculate business ratios**	2.1 **calculate business ratios**
2.2 check compliance with policy for product	2.2 manage profit and loss
2.3 check eligibility for credit transaction	2.2.1 **calculate sales**
2.4 analyze bank cashflow	2.2.2 calculate purchases
2.4.1 process turnover	2.2.3 calculate expenses
2.4.2 **calculate sales**	3 evaluate guarantor
2.4.3 calculate difference	3.1 analyze physical collateral
3 score applicant with altman-z model	3.2 check default probability
3.1 retrieve standard weights	
3.2 calculate **business ratios** and weighted score	

Table 5.4: Existing assets

5.2.2.3 Variation analysis

In this step, we first checked the duplication and variation levels in the existing business functions on graphs in Fig 5.7. Next, we checked those graphs with the the graphs of the required scope of functionality

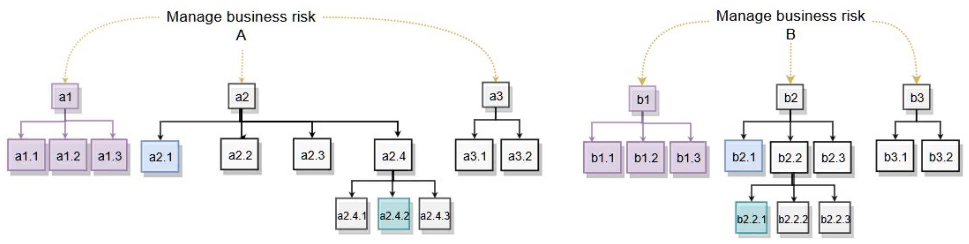


Figure 5.7: Similarities in existing functions

to look for existing assets that can be reused. For example, the functions *f1, f3.1, and f4.1* are found with a full match. However, for altman-z scoring *f2.1* different composition is found (See Fig 5.8). At this stage, we can draw the initial levels of granularity based on this analysis. At first, this will take into consideration found assets (units) and the maximum coarse-grained for those that are not fully matching (See Fig 5.9).

5.2.2.4 Service consolidation

After getting the initial levels, the functionalities *f3.2, f3.3, and f4.2* are still required. To maximize the reuse (f4.1), s6 can be created to implement f4.2. However, for the other two functions, the existing altman-z implementation needs a refactoring. Based on our reverse engineering activities in the various function areas, the ratios used for altman-z are standard in financial models and are re-implemented in various business components, including financial analysis and training as well as in the extracted business functions for cases *A & B*. Therefore, we create a separate service for them and refactor the existing a3.1 and a3.2 to include the operation in one service s5. Altman-z is a standard credit scoring model to predict whether an enterprise applying for a loan at a financial institution is of high risk of bankruptcy and is highly standard. Ratios can be taken out and *f3.2 & f3.3* are made as *f3.2'*. The refactoring efforts are discussed in the next section.

On the other hand, there are two choices for realizing f2.1 (See Fig 5.10):

1. Either we create a coarse-grained service **s2**, which will implement a2.4;
2. Or we create three fine-grained services **s2', s2", s2"'** to reuse the implementation of *b.2.2.1* which is the same as required *f2.1.2*.

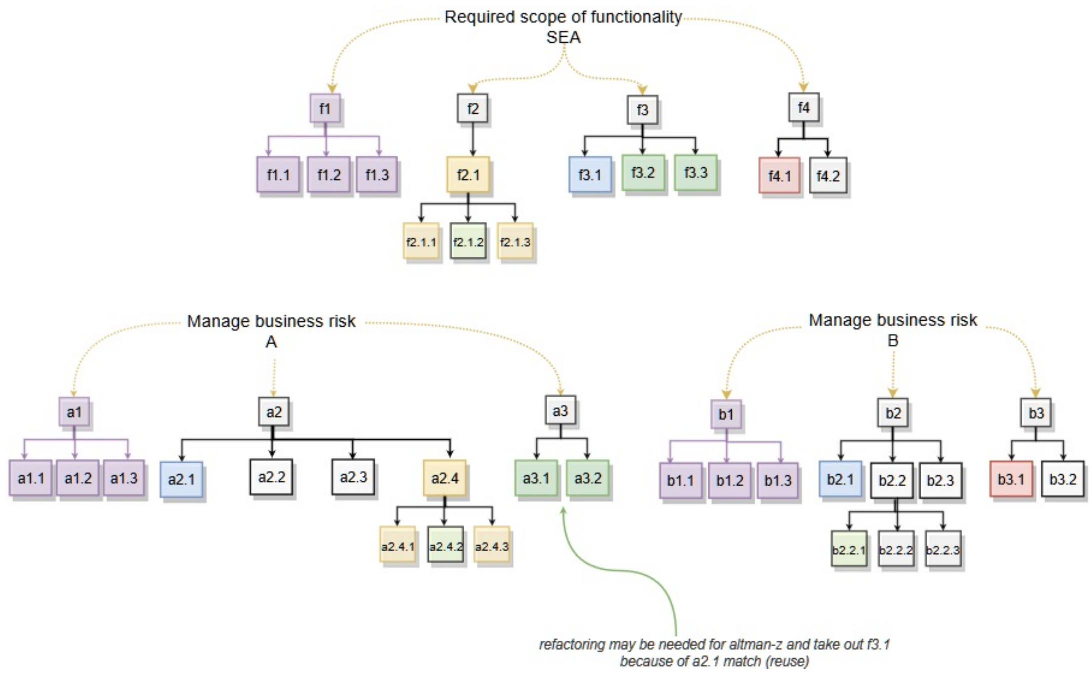


Figure 5.8: Variation in existing and required functions

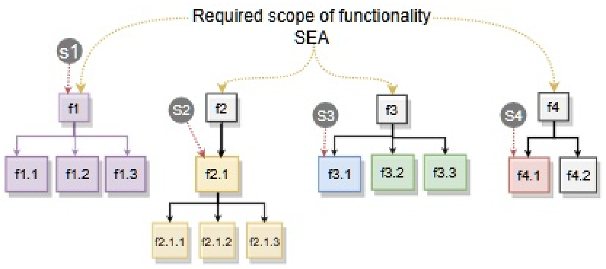


Figure 5.9: Levels of granularity: variation analysis result

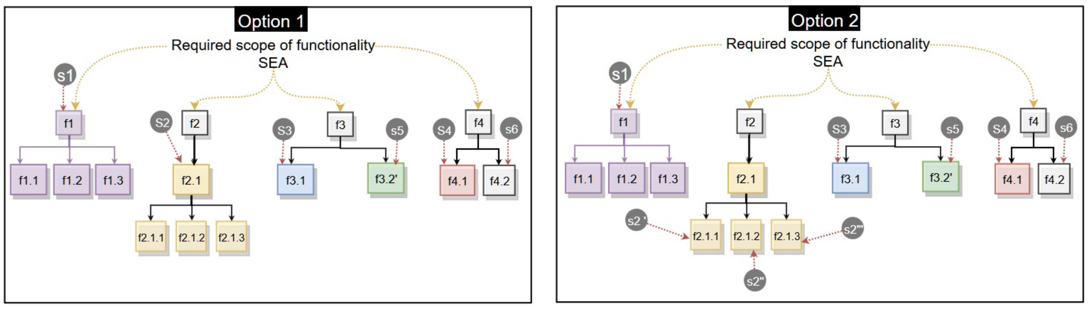


Figure 5.10: Service portfolio

5.2.2.5 Cost estimation

To size functions and services, we use so-called stachanov-point (stp), which is currently used by Stachanov in Jira task-logging to size development tasks and functionalities. This helps us estimate the size of new services to be implemented compared to the already implemented functionalities and spent efforts. The other cost rates and constants values are shown in Table 5.5. The rates[3] are provided per 1 stp based on the average rate for median Stachanov developers: In Tables 5.6 & 5.7 we present the sizing of both function units and identified services[4]. Given the refactoring size for s5 = 2 stp, we present the values of the granularity and cost metrics in Tables 5.7 and 5.8. Now, we can calculate

[3]A rate can also be in man-hour instead of a monetary unit.

[4]Sizes of existing functionalities were derived from analyzing 120 entry in Jira time-tracking. In the future, this sizing will be for SOA services instead.

Current cost parameters					
iRate	**iconst**	**scRate**	**scConst**	**mRate**	**exp**
30 €	25 €	20 €	10 €	25 €	1.20

Table 5.5: Input cost parameters

Function unit size							
Function	*Size(stp)*	**Function**	*Size(stp)*	**Function**	*Size(stp)*	**Function**	*Size(stp)*
f1.1	6	**f2.1.1**	6	**f3.1**	5	**f4.1**	8
f1.2	8	**f2.1.2**	6	**f3.2**	1	**f4.2**	4
f1.3	2	**f2.1.3**	6	**f3.3**	2		
Composition size							
f1	0	**f2**	2	**f3**	3	**f4**	3
High-level functions size							
f1	16	**f2, f2.1**	20, 18	**f3**	11	**f4**	15

Table 5.6: Function sizing

Service	S1	S2	S2'	S2"	S2"'	S3	S4	S5	S6	Total
Services size	16	18	6	6	6	5	8	2	4	53 (opt1-2)
Granularity size-metric	0	0	0.66	0.66	0.66	0.63	0.66	0.38	0.33	-
Granularity scope-metric	0	0	1	1	1	1	1	1	1	-
Implementation cost *opt1*	25	25	0	0	0	25	25	$30 * 2^{1.20}$	$30 * 4^{1.20}$	327 €
Implementation cost *opt2*	25	0	25	25	25	25	25	$30 * 2^{1.20}$	$30 * 4^{1.20}$	377 €

Table 5.7: Service sizing and implementation cost

Composition	Opt1: s2 (f2)	Opt2: s2', s2", s2"' (f2)	s3, s5 (f3)	s4, s6 (f4)	Total
Service composition cost	$20 * 2^{1.20} + 10$	$20 * 2^{1.20} + 10$	$3^{1.20}$	$20 * 3^{1.20} + 10$	282 €

Table 5.8: Service composition cost

the total costs for the identified portfolio P for either options to implement f2.1 and expose it as a service.

Total costs for option1: TC(P) = 327 + 282 = 609 €

Total costs for option2: TC(P) = 377 + 282 = 659 €

If we don't consider leveraging existing business functions and reusing implementation, one option is to identify services at the high-level business functions, the costs are shown in Table 5.9.

High-level business function	f1	f2	f3	f4	Total
Function size	16	20	11	15	62
Implementation cost	$30 * 16^{1.20} + 25$	$30 * 20^{1.20} + 25$	$30 * 11^{1.20} + 25$	$30 * 15^{1.20} + 25$	3330 €

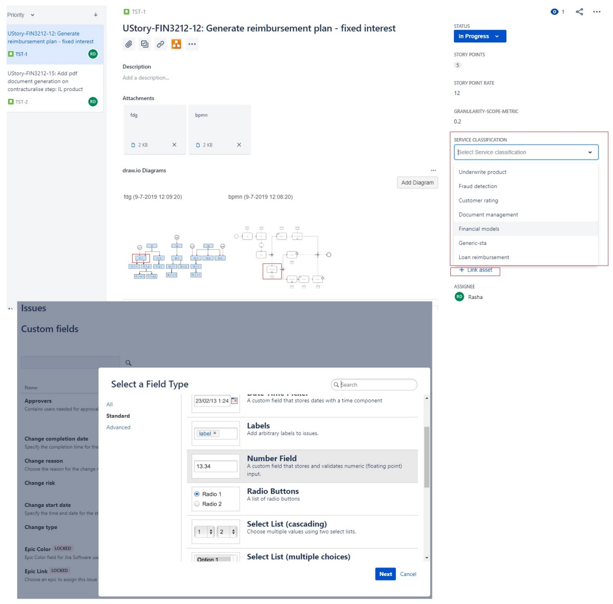
Table 5.9: Implementation cost: no use of existing code

5.2.2.6 Assessment and decision making

Since f2.1 is already implemented in case A, the cost of creating three separate services and orchestrating them instead of creating one coarse-grained service s2 to do the task is higher. However, since all those units already exist and f2.1.2 can implement another business function, the choice was for option2, so services can be reused to assemble other potential business functions.

Beside that, for s1, s3 and s4, there is an avoided cost from reusing existing assets. Mainly because the cost of exposing them as a service is much lower than the cost of creating a service and implementing the functionalities from scratch for the solution.

At the end of the decision iteration, we have established modelled business functionalities as services and list of services with their linked granularity values and costs. Of course, when the functional scope is implemented, those costs will be logged in jira and can be used for future cost estimation in next decision iteration and for monitoring the portfolio periodically. In current jira system among other options, there is already the possibility to add extra custom fields per ticket or release and configure it with numeric formulas to calculate the actual cost in respect to the original estimated costs (See Fig 5.11).


Priority
UStory-FIN3212-12: Generate reimbursement plan - fixed interest
TST-1
UStory-FIN3212-15: Add pdf document generation on contracturalise step: IL product
TST-2

TST-1
UStory-FIN3212-12: Generate reimbursement plan - fixed interest
Description
Add a description...
Attachments
fdg 2 KB
bpmn 2 KB
draw.io Diagrams
Add Diagram
fdg (9-7-2019 12:09:20)
bpmn (9-7-2019 12:08:20)

STATUS: In Progress
STORY POINTS: 5
STORY POINT RATE: 12
GRANULARITY-SCOPE-METRIC: 0.2
SERVICE CLASSIFICATION: Select Service classification
Underwrite product
Fraud detection
Customer rating
Document management
Financial models
Generic-sta
Loan reimbursement
+ Link asset
ASSIGNEE: Rasha

Issues
Custom fields
Name
Approvers
Contains users needed for approval
Change completion date
Specify the completion time for the
Change reason
Choose the reason for the change r
Change risk
Change start date
Specify the time and date for the st
Change type
Epic Color LOCKED
Epic Color field for Jira Software use
Epic Link LOCKED
Choose an epic to assign this issue

Select a Field Type
Search
All
Standard
Advanced
23/02/13 1:24 — A custom field that stores dates with a time component
Labels — Add arbitrary labels to issues.
Number Field — A custom field that stores and validates numeric (floating point) input.
Radio Buttons — A list of radio buttons
Select List (cascading) — Choose multiple values using two select lists.
Select List (multiple choices)
Next Cancel

Figure 5.11: Jira configuration

5.3 Repeating the application

We've demonstrated how the service identification and granularity decision approach works using a simple example from two delivered applications. Due to the nature of the business applications Stachanov develops for customers in banking and micro-finance industry to automate similar procedures, we repeated the application of this approach on four different applications to extract potential services. We started from the high-level business processes and BPMN models to extract distinct business activities. Those were aggregated based on their activity and invoked java file names to be compared. For example, for loan reimbursement activities, the decomposition of business functions was modelled in the four applications. The clone check revealed duplication but also differences mostly in the reimbursement plans calculations and functionalities due to the custom nature of interest rate plans and standards of each bank. To find proper boundaries between custom business logic and recurring logic with simple data type exchanged, we had to perform manual code analysis, consult the original requirement documents, conduct directed conversations with Stachanov staff due to the high complexity of the variations in banking reimbursement policies and interest rate plans.

By decomposing business functions in the distinct function areas of the grouped activities, medium and fine-grained functionalities were extracted that can be exposed as services or included as service operations to extend interface granularity which can facilitate implementing them in the future and having loose-coupled data models in service components. Some of the cases, the natural decomposition needed to be adjusted to set proper boundaries based on clones and data entities analysis. This was also due to the different coding patterns and variations in control flows. Some other cases were discarded as no enough models were there to do a comparison and because of the high cost of refactoring and service creation compared to the potential for reuse.

The extracted list of services list of extracted services from the platform and analyzed web applications is presented in Appendix B. We also included the main observations from the reverse engineering and code and variation analysis activities, which were reviewed by architects.

When developing SOA solutions, the more business services are assembled from service assets to customers, the more models can be compared which helps in taking optimized granularity decisions for maximizing reuse and assessing costs. It can also help in spotting problems and exploring potential business services that can be assembled from those assets. This also helps when new requirements come from customer to deconstruct part of a business service and combine other custom SOA services for the solution.

In Stachanov case, many delivered solutions include business functionalities for two or three different loan product types in the same bank. However, since requirements were iteratively delivered over the last two years and with no proper documentation or modelling for delivered functionalities and assigning development tasks to different people, it was hard to take cost-efficient design decision and prevent the damage to solution changeability and maintenance costs.

Chapter 6

Discussion

In this chapter, we connect all gained insights and We discuss how our developed solution is different from existing approaches contribution. We also answer the main research question and discuss how the approach can enhance decisions and coordination at the software company and enhance the alignment of IT and business.

6.1 Novelty of the approach

Based on our systematic literature review, we didn't find a service identification and modelling approach that provides an evolutionary overview of assets and how those assets contribute to deliver business value in already delivered solutions to enhance design decisions. Also, existing methods don't connect development activities and stakeholders with portfolio management activities and cost analysis in one solution. Those are usually separate processes and loops, which makes it hard to send feedback to each other in a seamless way and to not make bad design decisions for the sustainability of service portfolios. Also, coordinating between the various IT teams and also portfolio staff and business stakeholders requires extra efforts.

Development teams make daily design decisions based on available requirements to them and scoped work in backlogs. They evolve solutions and implement new assets with the continuous development and changes in requirements. Without an efficient leveraging to existing assets and without having proper knowledge about assets business capabilities when modelling requirements and identifying services, teams can take improper granularity decisions and create new services with low value while they increase future maintenance costs and affect service portfolio sustainability negatively.

The developed approach adds the development cost dimension to the decision-making process by integrating proper granularity and cost metrics in it. It also provides a shared modelling technique for required functionalities and already delivered ones to speed up leveraging existing assets and assembled business services in the design loop and take proper granularity decisions. This also allows learning from past deliveries and other teams.

The novel dimension is the self-learning dimension, by maintaining an up-to-date knowledge-base about developed assets, past efforts, and issues and success cases. Adopting our approach with the continuous development and delivery in Agile projects allows Agile teams to introduce optimal granularity decisions in consideration of development costs and the evolution of portfolios. Learning from past problems can also enhance decisions and reduce risks. The knowledge-base with the self-learning connects the monitoring and management of portfolios with Agile design activities. It continuously aligns software solutions and created SOA services with business needs, so they serve targeted customers and markets for a longer time, thus maintains higher sustainability for service portfolios. It also allows Agile teams to develop granularity patterns over time which can solve similar domain problems. This offers an efficient software lifecycle management solution to reduce risks, control costs, and maintain the profitability of available services rather than expanding assets in inventories, which adds more complexity and requires more efforts and resources to manage and optimize.

Development teams that can be organized around business domains and certain products can take time and cost-effective design decisions. Therefore, they are encouraged to make changes and release continuously and let the decision support process learn from its past support and their past mistakes and success.

6.2 Answering supporting research questions

i ***How to speed up leveraging existing assets and enhance the knowledge sharing across teams?***
Knowledge ownership and knowledge transfer are common problems, especially when various teams are working on different complex projects, and with the different type of information available to them. Strong cohesion and having a standard knowledge model for services assets can help development teams to understand the capabilities of those assets and access them to introduce changes or for reuse in implementing new business services and solutions. Therefore, it's important to maintain a standard description in service catalog and pipeline (planned services). Also, developing a shared modelling technique for both requirements and existing assets as they are assembled from different IT services and components can help development teams understand the business capabilities of existing assets when leveraging them in design loops. This also enhances the alignment between IT assets and business needs, which is essential for delivering suitable solutions to customers and allowing development teams to work independently. On the other hand, having a standard service catalog that describes features of business services helps business and marketing stakeholders do their job efficiently as well as allows to discover new services and take decisions to phase services out.
There is also the risk of people leaving the software company, which can be mitigated by enhancing the shared interpretation for the information about existing assets and delivered solutions and having standard procedures to manage the portfolio. In this context, standards such as FitSM can offer companies procedures and templates to manage offered business services and SLAs, and to set proper process and clear process ownership to coordinate IT and business stakeholders around delivering promised business solutions to customers. However, such standards should be complemented with proper documentation for delivered services and manage replacement, changes, or fixes to underlying IT services without increasing incidents and service downtime and decreasing the reliability of releases

ii ***How to continuously update the service identification and modelling strategies during the software lifecycle using the feedback from the portfolio evolution?***
With the continuous delivery, service modelling and decision pipeline and portfolio management for already deployed solutions and services should be connected so it can offer feedback to each other. This also allows to take rationale decisions in the decision pipeline based on a broader context. This can be achieved by having and maintaining an up-to-date and shared knowledge-base during the evolution of the portfolio to give an evolutionary overview about assets and their past efforts, along with suitable cost metrics to take cost-effective decisions. The knowledge-base which is continuously updating offers feedback to Agile teams over available assets that can be used to assemble the solution in the pipeline and past costs from previous cycles. This allows them to enhance their service identification and modelling strategies based on the current situation and take opportunities to reuse existing capabilities and even stay content with the evolution of the product they work on every iteration. The service and cost monitoring processes based on the maintained knowledge for assets and efforts in the knowledge-base makes the portfolio management more efficient and in line with the situation in the development pipeline. It also allows the portfolio management process to offer reliable feedback about problems and costs to management and to Agile teams to improve decisions and mitigate future risks. Therefore, the continuous learning and knowledge management connect development cycle and portfolio management cycle together and offers elaborate feedback to Agile teams automatically to update their strategies

iii ***How to optimize service granularity with consideration to development costs and portfolio management?***
Service granularity has more than the functionality as an important aspect. Other important aspects that can affect current and future costs are: coupling, genericity, flexibility, reusability as is or through composability, sourcing, and complexity. The choice of service granularity directly influences the ability for a service to be reused by various customers and combined with other services to realize various functionalities. However, leveraging and interpreting available assets and the ability to discover external services are key to maximize the reuse and reduce local development costs, as well as sustain the portfolio and reduce management costs. Reusing coarse-grained blocks to assemble a solution can be more beneficial in term of local costs. Building an SOA so-

lution or a business service from coarse-grained services also reduces composition costs compared to considering too many fine-grained services. Even though fine-grained services may offer a high flexibility in use, they often perform at the technical level rather than offer a high business value. Also, adding too many of those services to assemble a functionality may cause high complexity when changes are needed and therefore adds more change efforts. In all cases, the decision pipeline should consider an efficient method to estimate costs when comparing between granularity options before considering implementing services and reusing or extending existing ones.

To optimize service granularity, reusing existing services to reduce local development costs is not enough though. It is important when assets and portfolios evolve to engage the feedback from portfolio management activities about issues and increasing costs in the decision pipeline to support flexibility in change in both domain and technology as well as align solutions with customer needs and enhance decisions. Also, this feedback is important when improper boundaries between services are introduced, managing SLAs and assigning measurable KPIs to monitor deployed services and costs become hard. Therefore, there should be always a balance between efficient development cycles and the long-term impact of decisions on costs and sustainability. To achieve this balance, development teams should be offered supporting knowledge about past work and proper modelling and description of delivered assets to enhance reusability, but they also need to access information about past issues and sensitivity points before considering certain levels of granularity and reuse of assets. Also, when considering legacy systems, modelling services from those following their natural composition in those systems may result in extra costs and increased complexity. It is important to balance between reusing code as-is and optimizing future benefit and costs. Therefore, the ability to use a reliable cost estimation model in the decision pipeline is important to compare available options from an economic perspective. This is offered by having integrated indicators and efforts knowledge in the knowledge-base in our proposed solution. Those don't only support granularity decisions, they also support the portfolio management cycle

6.3 Answering the main research question

How to effectively manage the evolution of the service portfolio in agile software development?

A continuous alignment between business needs with IT assets is key for the success of any service provider and for returning high revenue and satisfying customers.

With the continuous development and delivery in Agile practices, experience is not enough to maintain a balance between development costs, efficient deliveries, and sustainability of service portfolios. Even though self-organized development teams should be competent to take proper design decisions for the product they develop. However, they receive requirements incrementally and have access to limited information in backlogs. Over time and without having enough knowledge about already created solutions and feedback from previous releases, design decisions that are taken every iteration can result in increased size of service inventories and assets with higher maintenance cost and lower reusability, as well as introduce different models and patterns in SOA solutions. Having too many assets is hard to manage and can also lead to a higher change cost and service disruption to customers on technology changes.

In such incremental development methods, it is important to manage the evolution of the service portfolio effectively and in the right direction to the market. Therefore, leveraging existing assets and modelling them in a similar way as business requirements can better align those IT assets with business needs and can help development teams learn from past deliveries as well as take proper design decisions to reuse some existing assets to reduce development cost without introducing future costs and affect portfolios negatively. Having proper knowledge-base in place with modelled assets and spent efforts doesn't only support Agile teams in taking proper and cost-effective granularity decisions, it connects Agile development activities with portfolio management activities through the shared and up-to-date knowledge, which allows learning from past to align assets with business needs and sustain portfolios. With the bounded time and resources per sprint and customer pressure, having such a reliable overview and cost estimation model when developing a new set of features is key to scope work and to not miss deadlines. It also supports setting proper investment plans to create new services for future customers, regardless of Agile projects.

When working on various IT complex solutions and with different Agile teams, knowledge sharing is about setting proper knowledge models and coordination processes to manage knowledge at all levels and create learning processes and collaborative success. The best knowledge about the capabilities of assets and pain points to avoid comes from already delivered solutions. With no proper knowledge models and feedback from the evolution of the portfolio in place, maintaining a suitable service portfolio and reducing maintenance and correction cost as well as staying competitive in the market become hard.

As a conclusion, to effectively manage the evolution of service portfolios with the continuous deliveries and changes, mastery and having proper knowledge sharing approach for development pipeline and portfolio management with lightweight service portfolio observers such as costs to offer feedback to development cycles is important to continuously maintain suitable service assets for business and help Agile teams work independently without lowering sustainability and increasing costs. This way, when solutions evolve, the balance between development cost reduction, future maintenance costs, and service portfolio sustainability can be maintained.

6.4 Conway's law

Even though, Melvin Conway's paper [7] was originally rejected by Harvard Business School due to the lack of evidence on the formed hypothesis, yet his hypothesis became popular and was named Conway's law. This law states that:

"Any organization that designs a system (defined broadly) will inevitably produce a design whose structure is a copy of the organization's communication structure."

For the last few years, companies started to understand this relationship between the coordination and organization of the development teams and the structure of the products they produce. This is the same when developing SOA products and slicing functional requirements and identifying services. Companies such as Netflix and Amazon structured themselves around small and independent Agile teams that can manage own life cycle of the services they create with a great degree of autonomy. Such services with their independent concerns can change and evolve separately from one another, resulting in the ability to integrate and push changes to production much faster and revert them back if needed. It also promotes continuous learning to integrate services due to the continuous delivery and feedback [22]. According to Conway's law, if those companies had adopted less and larger teams, the more monolithic systems and coarse-grained services that are hard to maintain and or won't be reused would have emerged, which may not have given teams the same ability to adapt to change and experiment with customers. While companies with too many small teams and with different attitudes and competency levels working on similar products may produce a lot of fine-grained services that may overlap, which can increase maintenance cost and complexity in term of responding to change and increase inventories size and future costs.

If we take this law, a company which has a vertical organizational hierarchy, where IT teams receive business needs iteratively in agile can evolve assets that don't align well with business needs.

Therefore, for companies willing to give autonomy to their competent development teams, our solution can support them in managing the evolution of their service portfolios by organizing their resources around domain products and maintaining a proper knowledge modelling for available assets based on their delivered business value and capabilities. This can help teams stay longer with such products and enhance design decisions over time.

6.5 Promoting long-term decisions and collaboration

Software applications should be thought of as products or set of services that will live in production and evolve for long time to fulfill customer needs. At the development team level, applications are often looked at as projects or temporary tasks with a set of requirements rather than a long-lasting set of services and *next* set of features and enhancement to fulfill those needs. This can promote more short-term thinking and fixes and less business-oriented culture.

When having multiple development teams or developers working on various products, developers should still be able to work independently and evolve solutions to stay reliable for a long time. Therefore, they should be offered the means to efficiently enhance and reflect on design weaknesses and can also help them learn from others and follow their success. This can also help when new team members or junior developers come in. As a result, both the sustainability of SOA solutions and the service portfolio can be maintained. This can reduce the accumulative costs from short-term solutions that may become costlier to pay as time goes on.

6.6 Software Architectural Erosion

Software architectural erosion is a common problem in Agile projects, which may cause higher maintenance cost, component complexity, and erosion of quality attributes that are important to the software company in meeting their SLAs with customers. Introducing new services with improper levels of granularity or introducing changes to services that contradicts with the original goals of those with no proper access to assets or evaluation of such decisions in long term can also lead to such erosion. In such case, re-engineering or refactoring activities should take place to services and solutions, which is not a trivial and error-free task, especially when the company has already adopted SOA for some time and many services are already used in assembling solutions to existing customers.

Preventing such cases is less costly, which is often done by setting pattern monitoring and maintaining up-to-date architecture documentation for development teams, as well as providing code and pattern generation tools. Our methodology can work as an early erosion reversion tool to keep the architecture of SOA solutions highly resilient to change, while it works as an erosion preventing tool in less complex domain problems. It is important however to set dedicated resources and clear process ownership for maintaining the knowledge-base.

6.7 Perfectionism pitfalls

Existing service modelling methods can be good for today's needs. However, with the continual change in requirements and with new products to be implemented in the development pipeline, the question remands, shall we do it right or do it as soon as possible. How to find the balance between delivering efficiently with creating and using services that don't introduce future costs. This contradictory is always a straggle by companies that provide services to customers, not only software companies. Perfectionism can hurt as solutions of today don't necessarily fit tomorrow's problems. There is always something else to tweak and trade-offs to make. The key is to continually maintain the balance between all factors rather than taking too long to decide on how to move from requirements to services. The role of our approach is to allow all stakeholders to take decisions quickly while enhancing their awareness of business intends and assets capabilities and estimate costs. The cost metrics don't only allow to estimate local costs, they work as global observers in the service portfolio management solution which raise predictive alerts for the company to maintain this balance. If long-term observers are introduced to the portfolio management such as the suggested ones in Section 4.4, the key is to not overdo it and to set and balance those observers, so productivity and innovation don't decrease.

6.8 Threads to validity

Especially in the context of monitoring costs and for maintaining the modelled business services from SOA services and service components in the catalog, the company should define clear procedures and dedicated resources for those tasks. Keeping the description and models of existing assets up to date speeds up leveraging to those to cut costs, but it also reduces risks of members absence and leaving. Also, the portfolio management cycle may involve different resources to cover finance, risks, and customer satisfaction aspects. Moreover, standard procedures should be defined to introduce changes to services and to take services out of the catalog as part of the portfolio management cycle. The knowledge in our knowledge-base becomes an input for those procedures and can accelerate them. However, the procedures themselves should be reviewed periodically as well, to make sure that they are efficient enough.

Chapter 7

Support in different context

In this chapter, we discuss how this methodology can be used to support different concepts and advancements, such as Evolutionary Architecture and facilitates the experimentation in new markets and with new technologies to transform market demands into real software products with approaches such as HDD in a risk-aware manner.

7.1 Evolutionary Architecture

"An evolutionary architecture supports guided, incremental change as a first principle across multiple dimensions." [25].

The concept of EA aims to maintains changeable components and a good coupling between those components to allow companies to evolve solutions on requirements changes and to experiment and revert changes back without causing other parts of the solution to fail or causing high service disruptions to customer. This is also important when there is a need to change the technology or upgrade a certain platform.

With agile methods, when systems are changed and rolled out again and again, systems should still be reliable, secure, and don't break. Even though short feedback loops can guide teams in introducing changes and help them learn from problems to enhance their decisions, Agile doesn't guarantee both business and software agility.

To guide teams in building a SOA solution architecture that is resilient to change and allow for reverting changes with less pain, leveraging functionalities from previous cycles and understanding their capabilities is essential. Also, the small decisions that are made over time must be looked at periodically to see if they were good enough for software sustainability and the ability to evolve. Here comes the role of the portfolio management and cost monitoring.

EA principle "If it hurts, do it more often" needs to be enabled by having teams organized around different products on which they can work for a longer time and learn from their continuous delivery and feedback. The pain can however be reduced when speeding up leveraging assets and understanding their business capabilities based on past releases. Our solution offers software companies a methodology to enable and accelerate this learning process to reduce this pain and guide Agile teams in developing an Evolutionary Architecture for SOA solutions.

7.2 Hypothesis-Driven Development

"The key outcome of an experimental approach is measurable evidence and learning"

When businesses have ideas for new services and solutions with a vision about the suitable capabilities, it is hard to quickly create such services without iterative exploring different options and checking the outcome based on market feedback. Hypothesis-Driven Development (HDD) is a practice which brings the scientific method to the software development process, especially when the software company would like to target new markets or business domains. Unlike markets in which the company has already delivered many solutions to tackle domain problems, in this situation, there are a lot of risks and uncertainties on how to reach such domains and how to develop services and evaluate their benefit. Instead of being stuck with abstract practices and general recommendations, companies use HDD concept to form hypotheses and perform experiments to test the hypotheses, which are expected to achieve a certain outcome for the end-user. Compared to TDD, HDD focuses on testing what the problem the business is trying to solve before working on concrete solutions [1]. However, it is still a new concept which needs an enabler approach.

Even though combining practices such as HDD and the continuous delivery allows companies to amplify learning and accelerate experimentation in new markets, to innovate at low risk and cost, it is important to be able to generate proper modular and revert changes to check other options and go with small bets to market for each iteration. Using our approach with proper indicators allows companies to push small business services to markets and observe how they are welcomed and whether they deliver value before extending those services. It also allows them to learn from past success and help their IT teams create those small bets and push them quickly to market to hear feedback and experiment with other options if needed.

7.3 Complementing Lean in Agile development

The concept of lean management was first introduced by Toyota business to minimize the waste of their manufacturing without sacrificing productivity. However, it soon became a complete philosophy of management and thinking and yet design method that can be applied to any business and production process, including software development. Lean is deemed to maximize customer value while minimizing "waste", or simply creating more value for customer with fewer resources, human efforts, time, or space. A lean organization on the other hand aims to provide the highest value to customers by a perfect creation process that has "ideally" zero waste, This sounds too optimistic though. To be close to achieving such a goal, the organization should transform its thinking for optimizing separate assets, technologies, and vertical departments to optimizing the flow of the organization products and shared assets. This requires a complete transformation of how an organization conducts its business and a supportive approach achieve those goals gradually.

Our proposed solution can help Agile teams to deliver faster and learn from the repeatable development process to generate value and reduce errors. Therefore, if we look at lean principles in Agile projects published by Meyer's book [19], our solution contributes to the following principles:

- Reduce waste: discovering potential services for reuse over redundancies and extra services that don't deliver value to customers and increases complexity and costs
- Amplify learning and create knowledge: learning from experience is good, however, to make it more effective, the shared knowledge models about past decisions, assets, and efforts can amplify learning. The knowledge is gained usually by doing the work, not by seeking perfect solutions. Applying our solution with the continuous development and delivery can make the learning process more effective
- Deliver as fast as possible by managing flows and taking time-efficient design decisions
- Establish pulls: however, companies that spend time working Agile won't necessarily reach assets agility without considering an integrated architectural decision approach that promotes patterns, standards, continuous assessment of assets and impact of work on business
- See the whole not the local: this is key for managing the evolution of service portfolios. Creating services for each set of scoped features in backlogs without carefully analyzing the past and leveraging assets can result in creating services that have a negative impact on business in the long term

Chapter 8

Conclusions and future work

8.1 Conclusions

To help software companies manage the evolution of their service portfolios in Agile development methods, we proposed a novel self-learning solution for effective service modelling and portfolio management. The solution provides a knowledge-graph based modelling and analysis techniques of already delivered business services and new required business functionalities in Agile development cycle to enhance the reusability of existing assets and avoid increasing maintenance efforts and the size of service inventories. It also integrates granularity and cost metrics and leveraging spent efforts from previous development cycles in the decision pipeline to take cost-effective granularity decisions.

With the continuous development and delivery in Agile, leveraging delivered modelled assets as services and past development efforts from the maintained knowledge-base in the development cycle and monitoring spent efforts and modelled services in the same knowledge-base in the portfolio management cycle make the proposed solution work as a self-learning design support tool for development teams to take proper design decisions based on learning from past decisions and issues, which can enhance the alignment of available IT assets in the portfolio and business needs. This self-learning dimension allows for enhancing the maturity level of design choices over time and allows software companies to have high business agility and maintain profitability in today's challenging market and with the continuously emerging technological advancements. Our approach, unlike existing approaches, connects the Agile development cycle with the portfolio management cycle and allow them to give feedback to each other efficiently for evolving the portfolio in the right direction for the market. Thanks to the shared knowledge-base between those cycles that offers an up-to-date overview of costs and portfolio evolution.

8.2 Future work

A possible future direction would be to build the knowledge-base part which includes the directed knowledge graphs over existing business services and functionalities using Artificial Intelligence (AI) capabilities. This would allow us to use AI algorithms to automate the service identification and consolidation steps on those graphs in the decision pipeline of new requirements. Similarities and variation levels can be detected based on the distance between function units on graphs and labelling similarities to help designers discover building blocks that can be reused in required business functions and avoid redundant efforts. This also allows to reduce the manual work and speed up the decision making process.

Another research direction is to research the application of our solution in managing the evolution of service portfolio for ENVRIplus project [1] [35]. The project aims to provide common solutions to the shared challenges of various Environmental and Earth System Research Infrastructures (RIs) in their efforts to deliver new services for science and society. It brings together existing projects and networks with technical specialist partners to create a more coherent, interdisciplinary and interoperable cluster of RIs across Europe. RIs usually offer datasets, and key platforms and tools to allow researchers to eectively engage with the available data and their own scientific field and areas of specialty. ENVRIplus has to host different RIs and should contribute to a multidisciplinary Earth system science across the traditional scientific fields. Such cooperation will help avoid fragmentation and duplication of efforts and make the European RI products easier to access. Our solution can help to develop a knowledge-base

[1]ENVRIplus Service Portfolio: *https://wiki.envri.eu/display/EC/ENVRIplus+Service+Portfolios*

with common semantic to offer access to information about research design and data resources that are made available by one of ENVRI community contributors. The knowledge-base can map environmental RIs and their technologies, and support comparison and analysis of RI designs to avoid duplication of efforts and enhance the discovery of data and tools. In this regard, the decision making concerns the data granularity, namely, the size of data of each node in the knowledge-graph that should be transferred to other nodes. The sizing metric should measure the data size while the cost metric can be replaced by a suitable metric to measure the transfer time of the data between different nodes. Those two indicators can help designers take proper decisions to use such resources and avoid causing an inadequate performance.

Acknowledgements

First of all, I would like to express my sincere gratitude to my supervisor Dr. Zhao for the continuous support of my research and thesis writing, for his motivation, patience, and immense knowledge. His guidance helped me in all the time of the research. I could not have imagined having a better advisor for this research project.

Besides my supervisor, I would like to thank the rest of the masters committee, for their encouragement and insightful comments during my study. I would also like to express my sincere gratitude to Stachanov staff, for giving me the opportunity to conduct my research project at their office in Amsterdam and for motivating me during my research.

Last but not the least, I'm thankful for my family in Paris for supporting me spiritually throughout writing my thesis and for their understanding for the busy time I've had which prevented me from being there for important family moments, and for my best friend Noucha for the warm support.

Bibliography

[1] How to implement hypothesis-driven development. https://www.thoughtworks.com/insights/blog/how-implement-hypothesis-driven-development, 2014.

[2] S. Overhage A. Albani and D. Birkmeier. Towards a systematic method for identifying business components. *Component-Based Software Engineering. CBSE 2008. Lecture Notes in Computer Science*, 5282:262–277, 2008.

[3] A. Allam T. Abdollah S. Ganapathy A. Arsanjani, S. Ghosh and K. Holley. Soma: A method for developing service-oriented solutions. *IBM Systems Journal*, 47:377–396, 2008.

[4] G. Aagesen and J. Krogstie. Bpmn 2.0 for modeling business processes. *Handbook on Business Process Management 1: Introduction, Methods, and Information Systems*, pages 219–250, 2015.

[5] D. Budgen P. Brereton M. Turner M. Niazi B. Kitchenham, R. Pretorius and S. Linkman. Systematic literature reviews in software engineering – a tertiary study. *Information and Software Technology*, 52:792–805, 2010.

[6] J. Bloomberg and R. Schmelzer. *The Agile Architecture Revolution: How Cloud Computing, Rest-Based SOA, and Mobile Computing are Changing Enterprise IT.* John Wiley & Sons, 2013.

[7] M. Conway. How do committees invent. 1967.

[8] F.Schlosser D. Beimborn, N.Joachim and B.Streicher. The role of it/business alignment for achieving soa business value - proposing a research model. 2009.

[9] R. Daigneau. Service design patterns: Fundamental design solutions for soap/wsdl and restful web services. 2011.

[10] N. Çizmeli E. Ungan and O. Demirörs. Comparison of functional size based estimation and story points, based on effort estimation effectiveness in scrum projects. *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 77–80, 2014.

[11] T. G. J. Schepers et al. A lifecycle approach to soa governance. proceedings of the 2008 acm symposium on applied computing. pages 1055–1061, 2008.

[12] M. FLYNN. Fit, lightweight and lean - fitsm, 2018.

[13] J. Geetha and T. Karthikeyan. A tool for measuring soa service granularity. 2012.

[14] J. Miller J. Arnold J. Cardoso, A. Sheth and K. Kochut. Quality of service for workflows and web service processes. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(3):281–308, 2004.

[15] R. Heckel J. Hausmann and M. Lohmann. Model-based discovery of web services. pages 324– 331, 2004.

[16] Y. Wu J. Jiang and G. Yang. Making service granularity right: An assistant approach based on business process analysis. *2011 Sixth Annual Chinagrid Conference*, pages 204–210, 2011.

[17] J. Qiao J. Liu, Z. Yu Xu and S. Lin. A defect prediction model for software based on service oriented architecture using expert cocomo. *2009 Chinese Control and Decision Conference*, pages 2591–2594, 2009.

[18] P. Kumar and R. Gupta. Dependency modeling of a soa based system through colored petri nets. *Journal of Computing and Information Technology*, 24:253–269, 2016.

[19] B. Meyer. . *Agile!: The Good, the Hype and the Ugly.* Springer; 2014 edition, 2014.

[20] Henry Mintzberg. Einführung unsere organisationswelt. 1991.

[21] V. Dwivedi N. Kulkarni. The role of service granularity in a successful soa realization a case study. *2008 IEEE Congress on Services - Part I*, pages 423–430, 2008.

[22] S. Newman. Demystifying conway's law. https://www.thoughtworks.com/insights/blog/demystifying-conways-law, 2014.

[23] H. Koziolek O. Zimmermann, L. Wegmann and T. Goldschmidt. Architectural decision guidance across projects - problem space modeling, decision backlog management and cloud computing knowledge. *2015 12th Working IEEE/IFIP Conference on Software Architecture*, pages 85–94, 2015.

[24] J. Koehler o. Zimmermann and F. Leymann. Architectural decision models as micro-methodology for service-oriented analysis and design. In *SEMSOA Workshop, Hannover, Germany*, 2007.

[25] R. Parsons P. Kua and N. Ford. *Building Evolutionary Architectures.* O'Reilly Media, Inc., 2017.

[26] P. Reldin and P. Sundling. Explaining soa service granularity. https://www.diva-portal.org/smash/get/diva2:23243/FULLTEXT01.pdf, 2007.

[27] L. Santillo. Seizing and sizing soa applications with cosmic function points. 2007.

[28] S.Izza and R.Imache. An approach to achieve it agility by combining soa with itsm. *IJITM*, 9:423–445, 2010.

[29] C. Steghuis. *Service Granularity in SOA Projects: A Trade-off Analysis.* PhD thesis, Business Information Technology, University of Twente, 2006.

[30] B. Tansey and E. Stroulia. Valuating software service development: Integrating cocomo ii and real options theory. *2007 First International Workshop on the Economics of Software and Computation*, pages 8–8, 2007.

[31] S. Watts. Itsm metrics kpi's for measuring success, 2017.

[32] WikiPedia Editors. Service oriented architecture. https://en.wikipedia.org/wiki/Service-oriented_architecture, 2019.

[33] G. Duysters Z. Li and V. Gilsing. The effect of knowledge decomposability on technological exploration in technological acquisitions. *Academy of Management Proceedings*, 2016:15882, 2016.

[34] D-c. Zhan Z. Wang and X. Xu. Stcim: a dynamic granularity oriented and stability based component identification method. *ACM SIGSOFT Software Engineering Notes*, 31:1–14, 2006.

[35] P. Martin Z. Zhao and B. Magagna. A definition of the envriplus semantic linking framework at conceptual and formal levels. http://www.envriplus.eu/wp-content/uploads/2015/08/D5.3-A-definition-of-the-ENVRIPLUS-Semantic-linking-framework-at-conceptual-and-formal-levels.pdf, 2018.

[36] L. Zhengyu. Essays on knowledge sourcing and technological capability : A knowledge structure perspective. Other publications tisem, Tilburg University, School of Economics and Management, 2016.

[37] O. Zimmermann. Architectural decision identification in architectural patterns. In *WICSA/ECSA Companion Volume*, 2012.

Acronyms

SOA	Service Oriented Architecture
ITSM	IT Service Management
BPM	Business Process Management
BPMN	Business Process Model and Notation
HDD	Hypothesis-Driven Development
DDD	Domain-Driven Design
SLAs	Service-Level Agreements
ROI	Return On Investment
KPIs	Key Performance Indicators
COTS	Commercial off-the-shelf
NPS	Net Promoter Score
SoC	Separation of Concerns
UML	Unified Modeling Language
WSDL	Web Services Description Language
IDL	Interactive Data Language
FDG	Function Dependency Graph
SLOC	Source Lines Of Code
EAF	Estimated Adjustment Factor
EAFs	Estimated Adjustment Factors
TDD	Test-Driven Development
SDLC	Software Development Life Cycle
DRY	Don't Repeat Yourself
RIs	Research Infrastructures
e-VRE	e-Infrastructures for Virtual Research Environments
AI	Artificial Intelligence
EA	Evolutionary Architecture

Appendix A

Stachanov background and studied case

A.1 Business process management and business tasks

A business process is a set of structured and ordered business activities and tasks to fulfill certain goals.

To develop and deliver a solution which enhances and automates business processes of a customer, software companies usually capture the various business activities the customer perform and their flow and organize and model them, then implement the underlying functionalities required in each activity. This allows the customer to work efficiently and transparently and reduce the amount of work required by automating simple tasks that may take valuable resources. Business process management solutions on the other hand usually automate the process from end-to-end.

Such a business process can be the procedure to process and grant a loan to applicant in the bank (see Fig A.1). The process includes activities and business functionalities related to processing the loan, taking decisions by various roles at the bank, generating contracts, and the various calculations and risk assessment algorithms. Automating such a procedure in a software solution allows every participating bank officer in it to work towards the end goal which is granting or rejecting the loan to the client in a pre-defined and structured way, take decisions when necessary, and check the status of the application and the historical actions on it.

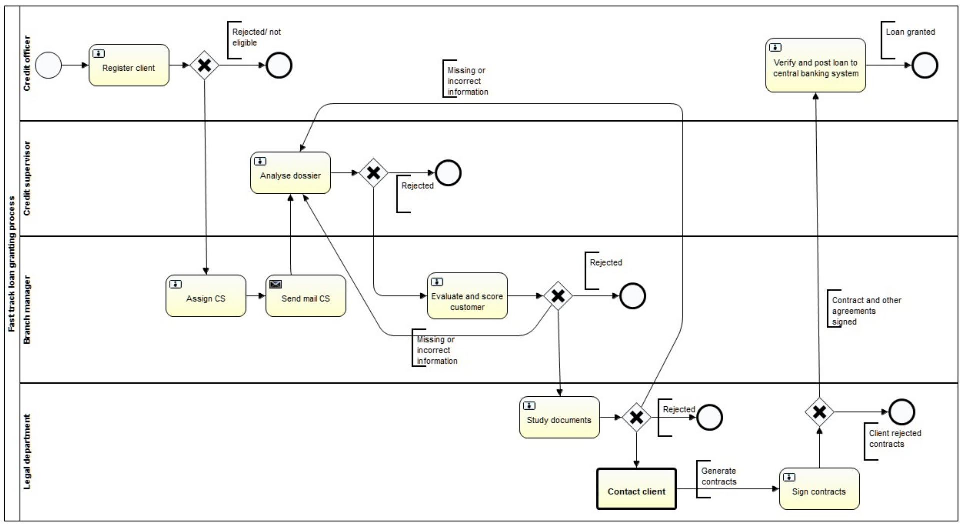


Figure A.1: High-level business process model for loan underwriting

A.2 Stachanov studied case

Since its foundation in 1997, Stachanov has been committed to assist its global clients in the rapidly expanding field of information technology and financial modeling techniques. Among others, Stachanov builds and delivers innovative and cost-effective business process management solutions to its global customers.

The company develops various web solutions to companies in the financial and banking sectors, such as automating business processes, assessing credit risk, and other banking simulation & training products. One of the most important platforms for business is an in-house BPM platform (so-called ARGOS 2.0). ARGOS helps the development team in developing web-applications to their customers. It currently works as set of functionalities that can offer developers utilities to rapidly build screens, include a complete workflow support to allow end-users take rapid decisions in the automated procedures and business processes, generate reports and manage user permissions on various levels such as pages and workflow tasks. The platform is a very powerful support tool to reduce time-to-market through the rapidly built screens, page and task authorities, and the workflow support.

Besides the platform itself which is a set of supporting functionalities for having the basic functionalities for a BPM web application, the company builds those web applications on demand to automate and manage loan and subsidy granting and repayment business processes for banks and financial entities, including managing related data, calculations and financial analysis, and modeling the overall business process. Those solutions allow banks to process loan applications and perform the flow of work in almost paperless, efficient and transparent way. This is very important for banks, so they can reduce the risks coming with granting loans and funding projects and start-ups through controlling and having a clear overview of all actions in the loan underwriting procedure for every single applicant, benefit from risk assessment and analysis functionalities with much less resources, and monitor the efficiency of the procedure itself.

A credit underwriting procedure usually includes many business activities and functionalities, involving various departments and stakeholders in the bank. It usually start from loan applicant intake and ends by disbursing approved loan amount or end of reimbursement and delinquency management. The process includes among others:

- Eligibility and pre-qualification checks in respect to applicant personal information and banking policies
- Application verification for application's correctness and completeness
- Risk analysis: risk and credit scoring based on standard or own scoring models at the bank. It can include trained credit scoring models from historical granted loans
- Financial analysis: current employment, bank statements, tax return, collateral, assets, history repayment behavior for existing customers
- Business activities analysis: balance, sales, purchases, expenses, cashflows, liabilities and equity, visiting business location
- Guarantees and insurance: guarantor business analysis and agreements
- Fraud detection algorithms
- Decisions and approvals: at various levels in the process
- Agreements and disbursement: preparing proposal, various calculations for reimbursement and repayment plans, monthly installment
- Document generation, reporting and statistics
- Operations with central banking systems and other subsystems

The detailed activities and underlying functionalities in each business task may vary from one loan product or loan plan to another in the bank, and from one bank to another. Each bank has a set of policies and loan packages to offer to its clients. Some products are directed towards small and medium enterprises, while others are individual loans, for personal or business use. There are also loan products that target groups to start a new business.

Stachanov capture the various activities in the business process and policies to refine them and construct a flow of activities and develop the required underlying functionalities and decision rules. As a result, stachanov delivers a web-application that allows end-users at the bank to use all types of business functionalities and calculations and complete their work efficiently. This also includes integrating outsourced services and banking services in the solution. On the other hand, Stachanov offer some of those functionalities and services separately or as part of banking simulation and risk assessment applications.

A.2.1 Used technology

In the following list, we give an overview of the main technology used in the current application landscape, including the main platform.

- Java: Java SE 7
- Grails: 2 (in 95% of the cases 2.4.4)
- IDE: GGTS 3.6.4.RELEASE with the platform Eclipse Luna SR1 (4.4.2) and groovy eclipse plugins
- Groovy compiler: 2.3.10, to compile both Groovy code and Groovy Servers Pages (GSP) code for "views". GSP is very similar to ASP and JSP, but far more flexible and intuitive
- Workflow engine: Grails Activiti plugin 5.22.0
- Workflow modelling: Activiti Eclipse BPMN 2.0 Designer
- Database engine: MySQL 5.7
- Spring security core: 2.0
- SOAP Webservice consumers: Grails CXF client plugin 2.1.2, which uses Apache CXF
- JavaScript
- JQuery: 1.11.1
- CSS

Appendix B

Application result in Stachanov case

B.1 Data model for demonstrated case

Fig B.1 shows the data entities and data hierarchy for the demonstrated case, subprocesses "Manage business risk".

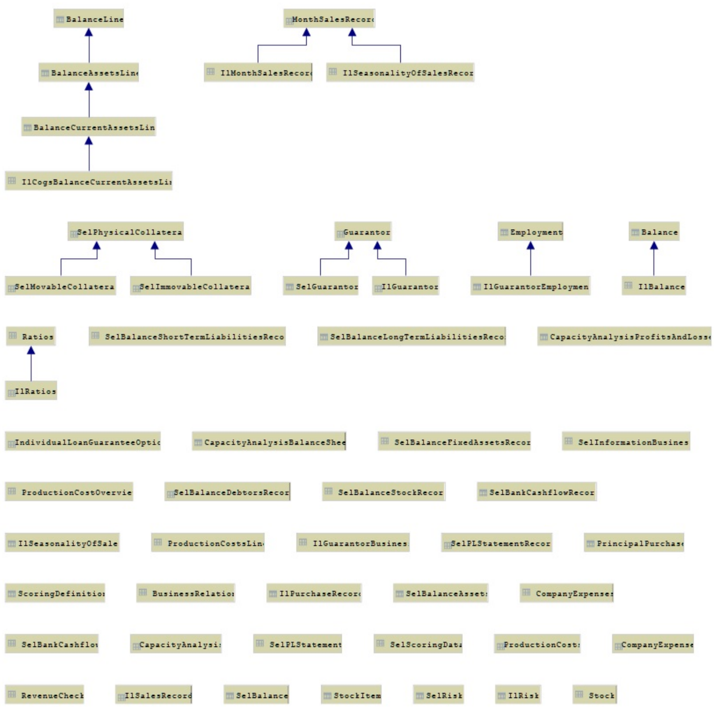


Figure B.1: Demonstration case data model

B.2 Candidate services for targeted market

Table B.1 shows the extracted subdomains and identified services as a result of reverse engineering the platform and the backend of the studied web applications.

B.3 Observations from reverse engineering and variation analysis

As a result of our reverse engineering and variation analysis activities, we extracted a list of main observations which can help Stachanov development teams in future design loops. Table B.2 presents this list.

Subdomain	Services
Standard credit scoring (trained models defaulter[a])	1. Calculate Bayesian score service 2. Calculate logistic regression service 3. Calculate expert score service: set weighs, retrieve weights, score applicant
Fraud detection	1. Benford detection service
Automatic notification (generic)	1. Mail service: *send mail, configure mail server, get service configuration* 2. SMS service: *send sms, set sms parameters, get sms parameters*
Credit risk assessment	1. Calculate Altman-z scoring service 2. Process historical cashflow service 3. Process projected cashflow service 4. Calculate business ratios service 5. Analyze collateral service: *analyze movable collateral, analyze inmovable collateral* 6. Analyze bank cashflow service 7. Analyze income statement service
Workflow support (generic)	1. Process definitions service: *add definition, delete definition, update definition, get definition by internal name, get list of candidate roles (task permission list), generate png diagram by internal name* 2. Process instance service (ACTIVITI engine integration): *add process instance, terminate process instance, get active tasks, get historical tasks, register decision, rollback decision, set assignee for task, get assignee task, delegate task, check deadline for task*
Document generation and management (generic)	1. Generate pdf document service (Open-sagres xml binding plugin) 2. Generate docx document service (Docx4J xml binding plugin) 3. Manage documents service: *check document is uploaded, delete document by business-object-id and domain-class-name*
Logging and audit (supporting)	1. User action logger service: *enable monitoring, log request, log response, create user request log, create log output string, map parameters for logging, retrieve session*
Standard loan application[b]	1. Manage loan application data service: *add loan application, delete loan application, get and set for: requested amount, requested term, goal, application date, contract date, first repayment date, contract end date, approved amount, and approved term*
Customer management	1. Personal information service: *add customer, delete customer, update attribute..* 2. Initial eligibility check service 3. Family composition service: *add family member, delete family member, update attribute, add family expenses, delete family expenses, update expense record, calculate total family expenses*

Table B.1: Candidate service portfolio

[a]A defaulter is somebody who fails to meet the legal obligations of a loan after it is granted. There are various scoring solutions that use continuously trained models from data of already disbursed loans to predict default probability in new loan applications, based on list of weighted attributes such as applicant age, gender, income, marital status, and number of years in business.

[b]Currently, a standard loan application logic and entity are encapsulated and found to manage the basic loan application data in existing systems. Managing the variable logic and data related to loan application is separated.

Observation	Problem	Reviewed by
Client registration and management functionalities as well as applicant management data model are added for different loan products and two different banks in one country	Loan applicant management should not be tight to certain loan product type, so it can be reused for different types within a bank or banks of one region or country	D. Morschhausen
Credit and business risk ratios as part of generic scoring models and custom financial models	There are many ratio algorithms and calculations, which are mostly standards and are also reimplemented as part of other business functionalities, such as banking simulation and business analysis activities as well as for group loans. They are also added to credit analysis and credit reporting services for different loan products (Small Enterprise, Individual, Group loan, Personal mortgages)	A. Koch
Some credit scoring models are highly standard and generic, but they are implemented in different applications	A model cannot be reused in new SOA application and it should be re-implemented. Therefore, they were included as separate coarse-grained services	A. Koch
Interest rate plans are part of reimbursement management functionalities	Interest rate plans vary between banks and within one bank. They should be included as separate custom services for SOA solutions, while reimbursement management functionalities can be reused across different loan products and banks as they can be orchestrated with interest rate services to use approved interest rate and the various amounts	F. ter Beek
Reimbursement management functionalities, including reimbursement plan generation are implemented in all cases	Similar to the above. If interest rate plans logic and data model are separated, new services can be created to be reused to assemble similar functionalities for various loan products and different banks. Also, re-implementing certain operations in reimbursement services (if needed) will be easier and less costly	F. ter Beek, D. Morschhausen
Automatic notification functionalities are implemented multiple times and they are rather generic	Replacing integrated plugins will result into the need to change the integration code in various applications	F. ter Beek
Different versions of COTS and plugins in different applications	No present reason for those variations. Possibly due to different developers working on different products. This makes it hard to patch and manage	F. ter Beek, D. Morschhausen

Table B.2: List of observations in Stachanov studied case