A method to leverage legacy Oracle Forms applications in an SOA

Jeroen Versteeg BSc

June 17, 2008

University of Twente
Enschede - The Netherlands

ORACLE®

Master thesis J. Versteeg Bsc

A method to leverage legacy Oracle Forms applications in an SOA

NAME	Jeroen Versteeg
STUDENT NUMBER	0015768
PLACE AND DATE	Utrecht, June 17, 2008
INSTITUTE	University of Twente, Enschede, The Netherlands
FACULTY	School of Management and Governance (SMG)
PROGRAMME	Business Information Technology (MBI)
COMPANY	Oracle Nederland B.V., De Meern
COMMITTEE	dr. M.L. Ponisio *1st university supervisor* Electrical Engineering, Mathematics and Computer Science (EEMCS)
	dr. M.E. Iacob *2nd university supervisor* School of Management and Governance (SMG)
	J.W. Sieben MBA *1st company supervisor* Consulting Manager, Oracle Consulting

Summary

In the last couple of years, Service Orientated Architecture (SOA) has gained a lot of momentum as a promising concept for IT systems. Vendors and organizations are adopting this technology in hope of creating more flexible and maintainable IT systems. While SOA looks like a very desirable architecture, most organizations do not start with a clean slate, but instead have made large investments in older technology, which usually does not integrate well with SOA. Many of these organizations face a dilemma with respect to the choice between keeping these outdated systems or replacing them with something new.

According to market research, thousands of organizations are facing this problem with respect to applications that have been developed using the Oracle Forms application development framework. Now that Oracle is adopting SOA for new products (and newer version of existing products), customers feel pressure to adopt SOA, and need to decide what to do with existing Forms applications.

While a lot of research has been done on replacing or re-engineering legacy systems, these approaches are often infeasible because of the costs and risks involved. Because these approaches have been developed using a technical viewpoint and hence do not take into account the particular business problems that need to be solved, they often cannot predict the actual business benefits the approach will deliver.

The decision support method presented in this thesis takes a business-oriented viewpoint and helps organizations find a solution which solves the business problems it is actually facing because of the system. To this end, the method directs participants to analyze why exactly the system does not fulfill a particular business requirement, and to find a solution that solves this problem. Since this research acknowledges that legacy applications should be abandoned in the medium to long term, the emphasis lies on finding solutions that either require little investments in the legacy system, or enable a gradual transition towards a more modern application.

The method has been validated using a small scale scenario based on a real case. While the findings show the method fills a gap in Oracle's Unified Method and is valid, more research is necessary to create a mature and well-tested method.

Acknowledgments

It is safe to say that this research would not have reached its goal without the help and support of a number people. First and foremost, I would like to thank my supervisors Maria Laura Ponisio and Maria Iacob for their feedback and their patience. I very much enjoyed discussing the direction of my thesis with them during our meetings in Enschede as well as via email. Maria Laura especially gave me guidance on which steps to take next, and perhaps most importantly, the encouragement to carry on when I needed it most.

Secondly, I would like to thank Remco Rosbergen and Jan-Willem Sieben, my supervisors at Oracle, for facilitating my project and introducing me to the right experts. In particular I would like to thank Jan Kettenis for taking the time to familiarize me with the OUM and to help develop my method, which is the primary contribution of this research.

Apart from the people who helped me directly with the research project and the thesis, there are number people who supported me personally during the project. A big thanks goes out to my family and friends, who supported me throughout the project and helped me cope with the personal issues I was facing at the time.

I would also like to acknowledge Gerben's patience with me during the time that I could contribute little to our company Asystance. It is because of his dedication and hard work that the company we started six years ago now provides us both with full-time employment and rewarding work.

Last but certainly not least I want to express my gratitude to all the friends I have made in Enschede, for making the six years I have spent there an enjoyable and unforgettable episode of my life.

Jeroen Versteeg, June 14, 2008

Table of Contents

Chapter 1: Introduction

In recent years, academia and commercial IT vendors have both described Service Oriented Architecture (SOA) as a desirable architecture for enterprise information systems (EIS) (see [Kontogiannis 2007], [Papazoglou 2005]). Promises of automatically configuring Business to Businesses (B2B) applications, lower maintenance costs, higher business agility and other radical improvements are widely spread, creating a sense of hype around the concept. Like many other technologies before it, SOA has raised great expectations about how it will revolutionize the enterprise IT landscape.

Academics and IT vendors agree that SOA is the best architecture to support current business drivers (see [Kontogiannis 2007]). If an organization would decide to set up its enterprise architecture from scratch, it could surely be wise to invest in SOA technology. The same holds true for investments in new applications, which should be “SOA ready”, even if the organization's underlying enterprise IT architecture is not yet.

If we can learn a lesson from previous hypes surrounding IT technology, we see that once the dust that was stirred up by the hype settles, expectations start to become more realistic, and pioneers report the real benefits of that technology.

The SOA movement is just in this phase at the moment of writing. While the various (and numerous) standards are still evolving, and research on the topic is continuing, vendors are already selling mature products, and some early adopters are already starting to reap the benefits of the new technology, while others face disappointing results [InfoWorld 2007].

The reality is that most enterprises already have large investments in applications designed around “old” architecture principles. These systems are called legacy systems because they are “technically obsolete mission critical elements of an organization's infrastructure – as they form the core of larger enterprises' business processes – but are too frail to modify and too important to discard.” [Papazoglou 2006]

Organizations want to have the best of both worlds: reaping the benefits of an SOA, while keeping long-term investments in legacy systems (which are mostly "well tested and tuned, and encapsulate considerable business expertise" [Cormella-Dorda 2000] *because* of these investments). Unfortunately, there is a true generation gap between not only the architectures, but also between the technologies used in legacy and state-of-the-art applications. So is it possible to bridge the gap somehow?

1.1 About this research

1.1.1 Problem statement

One of the companies facing this challenge is Oracle Corporation, which has recently made the strategic decision to adopt SOA standards for its entire product suite. The company already offers a comprehensive suite of SOA middleware products and is committed to making all of its applications SOA compatible in the near future.

Many of its customers have invested heavily in Oracle Forms, an older development framework for enterprise applications. These organizations now have large, monolithic applications based on legacy technology. While Oracle promises to support the technology for many years to come [ORACLE 2005], many business will find competitive pressure will force them to modernize their legacy IT applications to leverage the benefits that an SOA can offer long before vendor support for the current systems ends. These organizations need guidelines on how best to make the switch from old to new.

Oracle Forms and SOA are distinct technologies (see chapter 2), and no clear migration path is available to automatically move from Forms to SOA and Oracle's new development frameworks (such as the Java-based Application Development Framework[1]).

According to a recent report by Gartner ([Gartner 2007]), the move away from Forms can be delayed but not avoided entirely. Organizations should already start migrating towards modern application development technologies like Java or .NET, because Forms is "ill-positioned for next generation AD [application development] challenges". In conclusion, the "thousands" of applications built on Oracle Forms have to be migrated sooner or later.

But how can organizations "migrate" legacy systems to newer technology? According to Bisbal et al., "given the bewildering array of LISs [Legacy Information Systems] in operation and the problems they pose, it seems unlikely that a single generic migration method would be suitable for all systems" [Bisbal 1999].

Papazoglou et al. elaborate on their definition of "legacy" (quoted above) by stating: "Although there is strong corporate desire to replace legacy systems with modern technologies, the desire to replace these legacy systems is offset by real world constraints, such as massive long-term accumulated investments in the legacy systems and a significant loss of knowledge about the inner

1 See http://www.oracle.com/technology/products/adf/index.html

workings of these systems. The value of business logic in the legacy systems combined with the huge investments companies have already made in the development of their existing systems, are a powerful incentive to leverage these systems into modern business initiatives rather than replace them or recreate them with modern technologies." [Papazoglou 2006, p. 468]

1.1.2 Goal of this research

This thesis is based on a prescriptive design research. The goal of this research is to develop a method that helps in leveraging legacy applications built on Forms in an SOA. In this context, "leveraging" means any approach that delivers one or more of the benefits of SOA while not completely abandoning the investments that have been made in the legacy application(s). This is not to be confused with "maintenance", which is usually concerned with keeping a system running or correcting faults, without enhancing it to add value (see the definition on maintenance in section 4.1.1).

There are two reasons why SOA is the target architecture in the context of this research. The first is Oracle's choice to adopt SOA, which means that customers who wish to keep using Oracle's applications and middleware products also need to adopt SOA. The second reason is that a lot of organizations today wish to adopt SOA anyway. This research does not concern itself with the rationale behind this choice but instead accepts SOA as part of its context.

1.1.3 Research questions

The main research question is:

How can existing legacy IT applications be leveraged in a Service-Oriented Architecture, and which factors influence the choice between possible alternatives?

The first part of the question is vague and abstract because there are many fundamentally different ways to leverage legacy IT applications. Additionally, we have to find out exactly which alternatives are possible.

The second part acknowledges that some factors will guide or even dictate the choice between possible alternatives. Brooke and Ramage ([Brooke 2001]) state that business strategy must lead the evaluation of legacy systems. This project recognizes the need to take the business driver for change into account, and investigates the following factors:

- the organization's business driver to leverage the legacy application in an SOA
- the quality of the legacy application, and
- the organization's maturity with respect to the service-oriented businesses model and SOA technology

In order to be able to answer the main research question, it has to be split into sub questions. The answers to these questions will be compiled into a method (see [Hevner 2004]), which is the main contribution of this thesis (see section 1.1.5).

- Which business drivers are better supported by SOA than by legacy applications, and why?
- How can legacy IT applications be leveraged in an SOA to better support business drivers?

- Which requirements do leveraging approaches pose on an organization's SOA maturity and the legacy system's quality?
- How can an organization's SOA maturity be measured?
- Which quality attributes of legacy systems are relevant with respect to this thesis?
    - How can these quality attributes be measured?

1.1.4 Scope

This research focuses on a particular kind of legacy, namely monolithic applications built around Oracle Forms, a framework for building enterprise applications based on proprietary Oracle technology. This focus will also increase the utility of the research, since generic approaches have already been covered in the literature (e.g. [Lewis 2006], [Cormella-Dorda 2000], [Bergey 2001], [Lewis 2005], [Brooke 2001]), whereas the application of these general concepts in detail is lacking.

1.1.5 Contribution

The primary goal of the research project is to create a method that helps decision makers deal with the challenge mentioned above. The method takes an organization's business driver as a starting point and provides a recommendation on how to leverage the legacy application to meet business requirements.

Since Oracle Forms applications should be abandoned eventually ([Gartner 2007]), the alternatives for leveraging a legacy application considered in this thesis are short to medium term solutions aimed at fulfilling one particular business requirement.

1.1.6 Research approach

To answer the first research question, SOA and “legacy” (in particular Forms) need to be compared first. The next step is to identify which business drivers organizations currently face. With this knowledge in place, an answer to the first research question can be given.

To answer the second question, a literature study will be conducted to study and integrate knowledge on the topics of legacy system modernization and migration. The gathered knowledge will be extended and / or specialized by conducting interviews with Oracle experts in the Netherlands to integrate specialized knowledge on modernizing Oracle Forms applications.

After investigating which options are available to leverage Forms applications, interviews with Oracle experts will be conducted to identify the requirements they pose and research how these can be specified and measured.

Having now gathered all the necessary information, we develop the method described above.

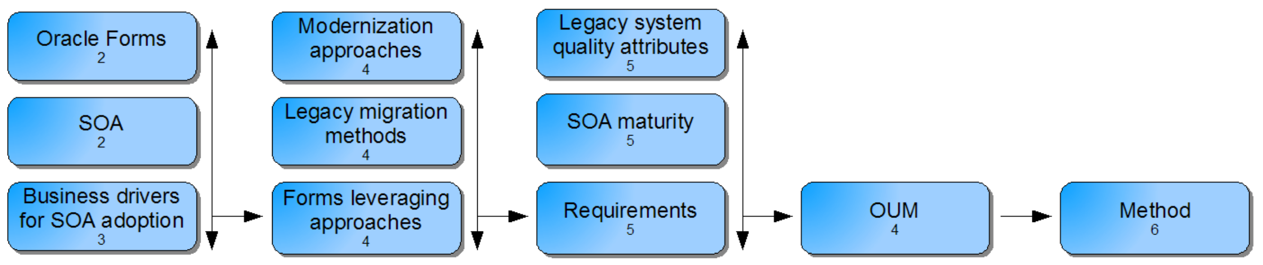

Figure 1.1: Research model

Figure 1.1 presents the research model for this approach.

1.1.7 Acceptance criteria and validation

The goal of this research is "to develop a method that helps in leveraging legacy applications built on Forms in an SOA". The method also has to take into account the "business driver for change". Together with the general guidelines for design science described in [Hevner 2004], these requirements form the basis of the validation criteria:

- **Innovativeness:** The method "must be *innovative*, solving a heretofore unsolved problem or solving a known problem in a more effective or efficient manner." [Hevner 2004]
- **Applicability:** The method needs to be applicable and complete. No steps or considerations relevant to applying the method should be ambiguous or missing.
- **Utility**: The method must be useful to Oracle and its customers. This criterion can be split into two sub-criteria.
  - The method has to fit both Oracle's existing methodologies and business strategy.
  - The method has to help the subject organization reach a decision about its short to medium term strategy regarding the legacy application.

1.2 Thesis structure

The remainder of the thesis is structured as follows:

Chapter 2 provides a detailed discourse of both service oriented and legacy architectures and compares the two in order to identify the key features that enable SOA to better support current business drivers as discussed in the third chapter.

Chapter 3 presents the business benefits to adopt SOA, based on a literature study. First, a list of common business drivers will be presented, followed by a discussion on how SOA is better equipped to support these drivers than legacy architectures.

Chapter 4 presents an overview of the literature about approaches and techniques that can be used to leverage or migrate legacy applications. The list of approaches is extended with those described and employed by Oracle specialists.

Chapter 5 discusses which requirements the approaches place on the legacy application and the organization, and which measurements can be made to evaluate which approaches are feasible.

Chapter 6 first presents related work on decision support methods found in the literature. The second section presents the method, which forms the main contribution of this thesis.

Chapter 7 contains the validation of the method based on a scenario.

Chapter 8 presents the conclusions and recommendations for Oracle and provides pointers for further research.

Appendix A described the Oracle Forms application development framework in detail.

Appendix B provides a primer for the technologies and terminology behind the "Service Oriented Architecture" (SOA) concept, and compares SOA architecture to (common) legacy architectures.

Appendix C lists the levels of Oracle's SOA maturity model and provides brief descriptions of each.

Chapter 2: Introduction to Oracle Forms and SOA

This chapter compares Oracle Forms to SOA technology and identifies key differences that make SOA the more desirable architecture.

2.1 Legacy systems in general

2.1.1 Definition

The term "legacy system" can be used for systems that are no longer supported (e.g. because the hardware they run on is no longer manufactured, or because it is very difficult to find skilled developers because the technology has become outdated), but this definition does not consider any implications (e.g. maintenance costs are high, or the system is resistant to change). In this thesis, Papazoglou's description of legacy is used, which was introduced in the first chapter and is reproduced here for convenience.

Legacy systems can be defined as systems which are "technically obsolete mission critical elements of an organization's infrastructure – as they form the core of larger enterprises' business processes – but are too frail to modify and too important to discard." [Papazoglou 2006]

Papazoglou goes on to state that "[t]he value of business logic in the legacy systems combined with the huge investments companies have already made in the development of their existing systems, are a powerful incentive to leverage these systems into modern business initiatives rather than replace them or recreate them with modern technologies."

This is exactly the situation that organizations with large systems built on Oracle Forms face. While the technology is still supported (and will be for some time, according to Oracle[2]), the characteristics of the framework make these systems difficult to change.

2.1.2 Architectures

Architectures of legacy systems often follow a single- or two-tier approach. A tier is a logical layer of a system. The idea behind this is that these layers operate independently of another and expose services to upper layers and consume services offered by lower layers (similar to the layers of the OSI networking model). Dividing applications into multiple tiers is considered a good practice, since it reduces coupling, thereby making it easier to change or replace one part of a system with little or no effect to the other parts (see appendix A for more). The notion of "tiered architectures" also helps to compare different architectures.

A two-tiered architecture usually consists of one tier that includes the presentation logic (which creates user interfaces) as well as the data processing logic, and a separate tier that houses the database, which is accessible through the network. A single-tier architecture combines these two and usually runs on mainframes. The main characteristic of these architectures is the low cohesion of the application's code. This makes them very time-consuming and difficult to maintain, since local changes can have global effects, requiring extensive analysis of the code before making changes and extensive testing after the changes have been made.

The client applications in two-tiered architectures are often called "fat clients" because they provide rich functionality by handling input validation and data processing locally, in contrast to "thin clients" which require servers to perform these tasks for them.

2.1.3 Problems

Most legacy architectures have several major drawbacks, as has been thoroughly discussed in the literature ([Britton 2004], [Papazoglou 2006], [Alonso 2004]). The most important problems are monolithic development and tight coupling, which both make maintenance complex and time-consuming, and use of proprietary standards, which hinders system integration and requires specialized development skills.

2.2 Oracle Forms

Oracle Forms is an application development framework for enterprise applications. It allows developers to quickly create applications using the developer suite. Applications built on Oracle Forms technology[3] vary greatly in size, from applications with only a few dozen "screens" to enormous systems consisting of several hundreds. Please refer to appendix A for a more detailed introduction to Oracle Forms.

Oracle Forms applications fit the general description of legacy systems quite well. They also fit the definition, since their architecture and other technical characteristics often make them difficult to

2 See http://www.oracle.com/technology/products/forms/pdf/10g/ToolsSOD.pdf
3 See http://www.oracle.com/tools/oracle_forms.html

modify. However, this is not necessarily true for every application. The maturity of the framework and the development tools do make experienced developers very productive.

2.3 Service Oriented Architecture

The Open Group defines SOA as "an architectural style that supports service orientation. Service orientation is a way of thinking in terms of services and service-based development and the outcomes of services. A service:

- Is a logical representation of a repeatable business activity that has a specified outcome (e.g., check customer credit; provide weather data, consolidate drilling reports)
- Is self-contained
- *May be* composed of other services
- Is a 'black box' to consumers of the service" [OPENGROUP 2006]

The term SOA is also used to refer to a set of standards, including SOAP, WSDL, UDDI, BPEL and others, which describe how services can be described, invoked, published and integrated.

Appendix B contains a detailed description of SOA.

2.4 Comparison

To understand the key differences between Oracle Forms and SOA, we will compare different aspects of each. We will compare commonly used technology, the division of the application among tiers and integration technology.

2.4.1 Standardization

For the last decade, the programming language of choice for enterprise application development is Java. The use of the language is so widespread that one could call it an industry standard. Sun, the creator of the language, has also recently opened the specifications and source code for the language, a move which will further increase compatibility between Java platforms.

However, in an SOA, which programming language is used to implement services shouldn't matter, since services can be considered black boxes that only have to expose an API using standards. So although the standardization of Java is good for lowering maintenance costs and other reasons, from an architecture point of view, it doesn't matter in which languages the services are implemented, and developers are free to chose different languages and even platforms (hardware, operating systems) as long as the APIs of the services use standards.

The situation is completely different in legacy applications. Each application can use different technologies, and integration and maintenance efforts have to consider these differences. The problem is even worse when the technology is proprietary, such as Oracle's PL/SQL.

The bottom line is that SOA is based on open standards, where the Oracle Forms framework relies on proprietary technology and standards. The former makes development easier and less time-consuming, since products from different vendors (*should*) work together and developers need no knowledge of proprietary standards and technology.

2.4.2 Separation of concerns

A good practice in software engineering and systems architecture is separation of concerns, which means that pieces of code that deal with closely related aspects should be isolated (or separated) from code that deals with other aspects. This separation is important because it helps to keep cohesion high and coupling low, which in turn makes modifications easier to carry out (see also section 2.1.2).

While separation of concerns has always been a best practice, the technology, architecture and standards used in many legacy applications do not enforce or even encourage strong separation of concerns.

Separation of concerns is an important aspect in SOA. Multi-tier architectures, object and service orientation, and modern development practices all advocate and encourage separation of concerns (although they do not force it). This makes it difficult to migrate legacy systems to SOA, since the architectures usually are fundamentally different.

This section discusses how separation of concerns is generally implemented and advocated in legacy applications and SOA.

Difference between application and business logic

Application and business logic are two similar concepts. In this thesis, application logic will refer to logic that is technical in nature and deals with processing information, while business logic is abstract and is concerned with codifying business policies and rules.

Examples of application logic include code that deals with converting one currency to another, or code that formats a machine-readable message to a human-readable form. Code that decides whether a purchase order needs management approval based on the total price reflects a business rule and hence is considered business logic.

However, due to the vague definitions, the coverage of both types of logic does overlap. The distinction is an important one to make though, since each type of logic is (usually) dealt with in different parts of an application in an SOA, where in most legacy systems, the distinction is not made, and both types of logic reside in the same places.

Application logic

In an SOA, application logic should be implemented in a separate “middle tier” (which lies between the data tier and the presentation tier) and decomposed into and exposed as services.

Legacy applications usually don't have a decomposition of logic into independent pieces that are small enough to match the granularity of services in an SOA. As discussed in appendix A, the applications also combine different kinds of logic (e.g. business and application logic) in the same pieces of code. This makes it even harder to reuse or change parts of these applications, since they are not isolated from the rest of the application.

To make functionality reusable, legacy application developers often make use of stored procedures, which are pieces of logic exposed through a function and embedded in the database. The advantage of using stored procedures is that logic can be reused and exposed using a strict API. Stored procedures can be wrapped and exposed as web services, which makes it easy to make the logic reusable in an SOA.

Businesses logic

In legacy applications, business rules are usually embedded in the code (and possibly stored procedures), together with the application and presentation logic. Depending of the definition of business logic, even database constraints (like foreign key constraints) might be considered business logic.

Just like application logic, business logic should be handled as a separate concern in an SOA. One of the obvious locations for business logic is a BPEL model, but there are alternatives, like (proprietary) rules engines (that expose their functionality as services) and database constraints.

The main point is that while businesses logic can be found in every layer of both architectures, it is usually considered a best-practice to manage it as a separate concern. This is usually the case in SOA applications, while in legacy systems, it is usually embedded in the application code and interwoven with the rest of the logic (e.g. presentation logic). Additionally, business logic is modeled explicitly in SOAs (see section 3.3.5), whereas legacy systems often contain business logic distributed among the application or depend on users knowing which actions to take and in which order.

Presentation logic

Legacy applications often use proprietary and platform-dependent interfacing technology, and, again, the logic is integrated with the rest of the logic, as described above. The technology is also usually designed for one particular client, for example desktop computers running Microsoft Windows or “dumb terminals” connected to mainframe systems.

In an SOA, the presentation tier is separated from the other tiers and able to create different interfaces for different clients. Platform-independent technologies like XHTML, CSS and XSLT are employed to improve portability. Of course, platform-specific and proprietary technology *can* still be used, but since the presentation tier is independent of the others, a new channel can be added quickly.

This makes it possible to use two user interfaces without duplicating the underlying application and business logic. For example, one interface could be designed for employees and provide all the available functionality, while a simpler and more user-friendly interface is provided to end customers.

As should be clear from the above discussion, separation of concerns is often lacking in legacy applications (which in turn might be one of the reason an application becomes legacy in the first place). SOA, on the other hand, enables and encourages separation of concerns.

2.4.3 Integration technology

Legacy system integration (for EAI or B2B efforts) can be achieved using different techniques, depending on the systems' (common) technology. One of the most common approaches for interfacing legacy systems is to directly access that system's database by running SQL queries on it. Another is to statically link systems' machine code at compile-time, which makes it impossible to change any aspect of the integration at runtime. Both techniques create tightly coupled systems and dependencies, which are considered bad practices because they make systems resistant to change.

A lot of alternatives have been developed (e.g. RPC, COM, CORBA - see [Britton 2004] and [Alonso 2004]) in the form of middleware solutions to solve these problems, but none has gained enough momentum to be widely deployed and used.

This is where web services promise fundamental improvements by providing interfacing technology that is based on mature and platform-independent technologies and open standards (see appendix B).

2.4.4 Summary

One of the design principles of SOA is clear separation of concerns. Applications are divided among multiple tiers, each of which are independent of one another. This makes the resulting architecture loosely coupled, and thus more flexible and easy to change when compared to most legacy systems, which often have almost the exact opposite properties. This means it would be beneficial if the method developed in this thesis helped in enforcing separation of concerns.

Legacy applications also mostly rely on proprietary and platform-dependent technologies, which creates a highly heterogeneous ecosystem that makes integration efforts difficult and costly. Most legacy systems also degenerate over time because the use of bad practices accumulates. This also negatively affects maintainability.

Table 2.1 provides a summary of the key differences between legacy and SOA based systems. It should be clear that the large differences make migrating legacy systems to SOA a difficult (and expensive) undertaking.

	Legacy (generalized)	**SOA**
Standardization	Proprietary technology and standards	Open standards
Architecture	Two-tiered; little emphasis on separation of concerns	Multi-tiered; separation of concerns strongly encouraged
Integration	Static bindings and point-to-point links create tightly coupled systems	Dynamic bindings and use of integration middleware allows loosely coupled systems

Table 2.1: Overview of differences between legacy and SOA based systems

Chapter 3: Business Drivers for SOA Adoption

If we want to understand why SOA is considered the optimal IT architecture for today's enterprises ([Kontogiannis 2007]), we have to take a look at the business drivers it supports, and why it supports these better than previous architectures. This chapter will answer both these questions. The first part of this chapter provides an overview of current business drivers and opportunities organizations face. Subsequently, the reasons why SOA supports these better are discussed, based on high-level advantages of SOA over previous architectures.

3.1 Introduction

As described in the introduction (see section 1.1.3), the business driver is considered the starting point or primary input for the method developed for this thesis. However, interviews with Oracle experts have shown that, while one or more particular business drivers are the starting point for most organizations to consider SOA adoption, it is only indirectly relevant to the choice between leveraging alternatives for the legacy system. What *is* directly relevant for this choice is the way in which the current system does not sufficiently support the business driver(s).

If we look at it this way, we see that particular business drivers pose different requirements on organizations' IT systems. It is not the driver itself that directly influences the choice between leveraging alternatives, but rather the requirement(s) this driver poses on the information systems. To make this distinction explicit, we can differentiate between business drivers and IT drivers.

This change in perspective with regard to the method's input has been taken into account in the method as presented in chapter 6.

The next section provides an overview of common business drivers. Since this research assumes SOA adoption as a given, justifying the rationale for SOA adoption lies outside of the scope of this research. Consequently, the purpose of this discussion is not to promote SOA adoption or to make the case for SOA, but rather to describe part of the context for this research.

Sections 3.3 and 3.4 evaluate how the requirements on IT systems identified in section 3.2 are supported by SOA. This leads to a set of advantages of SOA over legacy systems in general and Oracle Forms applications in particular, which is presented in section 3.5. It is this set that is used as input for the method as described in chapter 6.

Since the set of SOA advantages is derived from a non-comprehensive list of businesses drivers, we cannot be sure whether the set of advantages is comprehensive. However, by covering the most common business drivers and by validating the set of advantages in interviews with Oracle's expert business consultants, the general applicability of the method can be guaranteed. By ensuring the method's extendability, its usefulness can be further improved. Please refer to section 8.2 for a discussion of the final method's applicability and shortcomings.

3.2 Current business drivers and trends

As stated in the introduction (chapter 1), SOA is considered the de-facto standard for enterprise applications [Kontogiannis 2007]. Unfortunately, the academic literature is not as verbose about *why* this is so. It does list numerous technical advantages over previous (legacy) architectures (like reusability of code, separation of concerns, etc) [Alonso 2004], but it does not describe the necessity for these features from a business point-of-view, nor does it present solid empirical evidence to prove any of the claimed advantages are actually achieved. To paraphrase using the terminology of "Solution Selling" ([Bosworth 1994]), the literature *does* delve into the *features* and *advantages* of SOAs, but the discussion and proof of their *benefits* is lacking.

The industry literature *does* discuss benefits of adopting SOA. However, we have to be skeptical when investigating the claims, since the majority of the literature is made up of vendors' marketing messages wrapped in commercial white papers and industry magazines, both of which lack a neutral stance.

3.2.1 Frameworks

To take inventory of aspects that are important to organizations, this section investigates two frameworks to compile a list of the most common business drivers. This list is used in sections 3.3 and 3.4 to show how SOA supposedly supports these drivers better than other architectures.

Strategic alignment

According to Henderson and Venkatraman's model of strategic alignment ([Henderson 1993], see figure 3.1), strategic fit ("the need for any business strategy to address both external and internal domains") is critical for maximizing financial performance. The authors define an organization's external domain as "the business arena in which the firm competes", while its internal domain "is concerned with choices pertaining to the logic of the administrative structure (functional or divisional or matrix organization) and the specific rationale for the design and redesign of critical business processes".

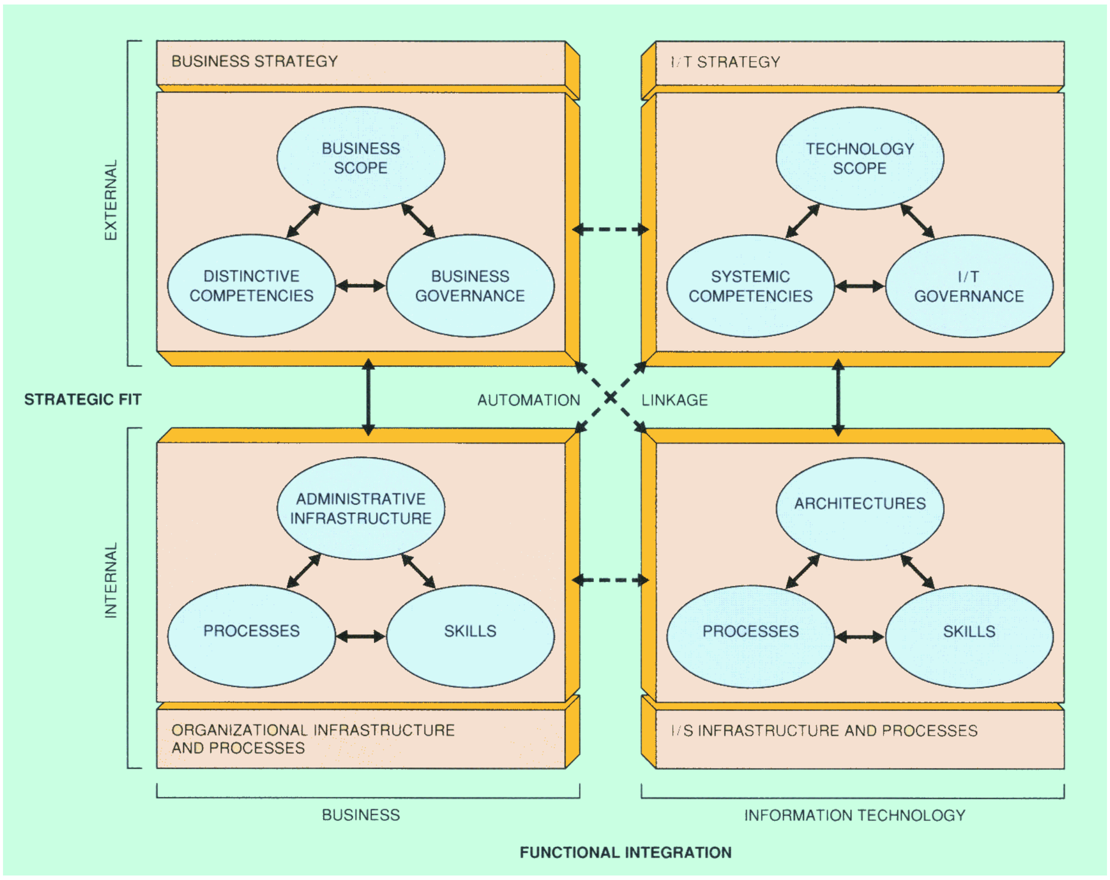

Figure 3.1: Strategic Alignment Model (source: [Henderson 1993])

On of the areas of the strategic alignment model that can be directly supported by IT architecture is systemic competencies, found in the I/T strategy quadrant of the model. Systemic competencies are "those attributes of I/T strategy (for example, system reliability, cost-performance levels, interconnectivity, flexibility) that could contribute positively to the creation of new business strategies or better support of existing business strategy".

Generic business strategies: Cost Leadership and Differentiation

To remain competitive in an increasingly globalized and fast-changing market, Ward and Peppard ([Ward 2003]) propose organizations have to adopt one of two generic business strategies (or both): the low-cost strategy and the differentiation strategy. However, "the majority of organizations have to follow a differentiation strategy, since, theoretically at least, only one company can have cost leadership of a product or service at any one time." [Ward 2003] The authors admit that these two options do not cover all available options and leave questions as to *how* these goals can be achieved.

They cite Treacy and Wiersma, who suggest that a significant range of the possibilities to achieve market leadership (using differentiation) can be represented by three paths (see figure 3.2):

1. "*Operational Excellence*—enabling products and services to be obtained reliably, easily and cost-effectively by customers. This implies a focus on business processes to outperform others and can deliver both low costs and consistent quality of customer satisfaction.

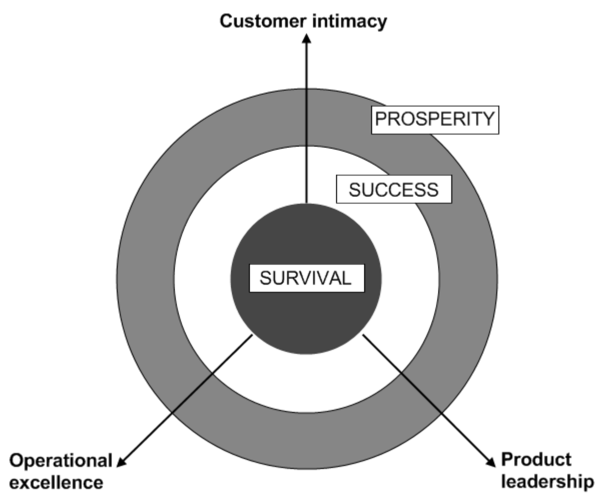

Figure 3.2: Dimensions of competency (source: [Ward 2003])

2. *Customer Intimacy*—targeting markets very precisely and tailoring products and services to the needs of particular customer groups. The purpose here is not just to 'satisfy' but to 'please' customers by understanding their needs and meeting them on every occasion.
3. *Product Leadership*—continuing product innovation meeting customers' needs. This implies not only creativity in developing new products and enhancing existing ones, but also astute market knowledge to ensure that they sell. The strategy involves delivering a continuous stream of new products and/or services, where what is new is valued by the customers." [Ward 2003]

Regardless of an organization's businesses strategy, financial drivers are always important and should be considered alongside any of the particular differentiation strategies mentioned above.

This section discusses which requirements these generic business strategies (including financial drivers) place on information systems. The focus lies on Operational Excellence and Financial drivers, since these two can be best supported by well-aligned IT.

3.2.2 Operational Excellence

From functional divisions to process orientation

For a long time now, organizations have been structured along functional lines, resulting in departments which group related tasks and the required expertise in order to gain efficiency. This view is challenged by the business process re-engineering movement, which seeks to focus on horizontal business processes as the basis for organizational design. "In short, processes are becoming the building blocks of organizations and seek to capture natural workflows." [Ward 2003] See figure 3.3 for a conceptual example.

Historically, departments have created information systems (often called "silos") that strictly follow the departments' boundaries and which often duplicate information required by different departments [Britton 2004].

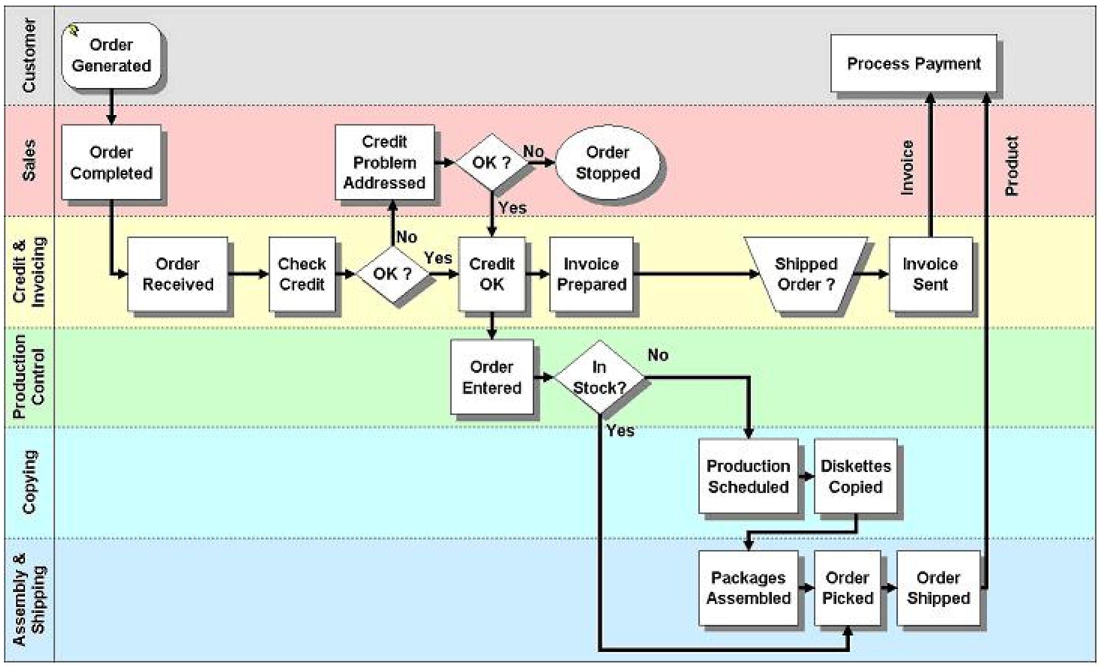

Figure 3.3: Business process spanning multiple functional departments (source: www.gemba.com)

Process orientation requires workflows that span different departments to be supported by different information systems. Hence, integration between each of these systems' data and functionality (termed Enterprise Application Integration or EAI) becomes a prerequisite. In light of the strategic alignment model, this relates to operational integration, "the link between organizational infrastructure and processes and I/S infrastructure and processes." [Henderson 1993]

In terms of Teacy and Wiersma's 'paths to market leadership', process orientation mostly supports Operational Excellence, but it can also support the other two paths.

Business Process Re-engineering and Business Process Management

To create efficient and effective business processes, organizations employ business process re-engineering (BPR) and business process management (BPM). BPR is defined by Hammer and Champy as "*the fundamental rethinking and radical redesign of business processes to achieve dramatic improvements in critical contemporary measures of performance, such as cost, quality, service, and speed.*" [Hammer 1993] BPM is often seen as the successor to BPR and focuses more on continuous evolution of processes instead of one-time radical changes and information technology support for processes.

Ward and Peppard state that it is not clear whether the role of information technology in BPR is the driver, an enabler or one of the means of implementation. They suggest that to answer this question, one first has to ask these two questions:

- How can business processes be transformed using IT (based on a full understanding of the capabilities of IT)?
- How can IT support business processes?

Without answering these questions in detail, we can say that, ideally, IT should suit BPR and BPM by supporting both aspects.

Business agility through componentization and service orientation

The environment businesses operate in is constantly changing. “Mergers, acquisitions, and the introduction of new technologies are examples of drivers for change in business environments. Business agility refers to the ability of an enterprise to thrive in a continuously changing and unpredictable environment.” [Elfatatry 2007] As environments change, businesses must reevaluate their core competences and create new competitive advantages.

Henderson and Venkatraman state that an organizations strategic fit is dynamic “The choices made by one business enterprise, or firm (if fundamentally strategic), will over time evoke imitative actions, which necessitate subsequent responses.” [Henderson 1993] The systemic competencies are an important aspect of I/T strategy when dealing with change and flexibility.

In order to be able to change fast, organizations need to be agile. Cherbakov et al. have recognized that “corporations are naturally becoming componentized.” [Cherbakov 2005] This means that instead of dividing business into business units and departments, they are divided into finer grained parts, called business components. Each one corresponds to one business function. This enables businesses to rapidly deconstruct and reconstruct to create new value nets on-demand. “In the on demand environment, the component-based firm links its components efficiently and seamlessly both internally and across the firm’s boundaries with best-of-breed components provided by external partners.” [Cherbakov 2005]

This seamless integration is achieved through service orientation. Business components provide on ore more unique services for consumption by other components. Interactions between components are governed by contracts that specify costs, service levels and other agreements.

According to Cherbakov et al., this creates on-demand businesses that can adapt to changing needs quickly. This is necessary to decrease the time-to-market for new products or services, which serves the Operational Excellence and Product Leadership paths.

Componentization businesses are ideally supported by componentized information systems which can be flexibly and easily integrated.

e-Business

Papazoglou and Ribbers define e-Business as “the application of information and communication technologies to conducting business” [Papazoglou 2006]. According to the authors, e-Business can lead to performance increases and cost reductions in various core business processes, among others:

- collaborative product development
- collaborative planning, forecasting and replenishment
- procurement order management
- operations and logistics

Papazoglou and Ribbers list eight requirements for e-Business, six of which are organizational requirements and two involve IT. These latter two are “align business organizations with a flexible IT architecture” and “establish ubiquity within standards”, both of which are well supported by SOA.

3.2.3 Customer Intimacy

Providing above-average service to customers requires efficient business processes (see above) and integration of information systems that store different data.

Another way of improving customer intimacy is to provide self-service applications to customers. These can be used around the clock and from anywhere in the world, offering flexibility and freedom. This enables customers to quickly file common requests or view personalized information, for example. Not only does this increase customer satisfaction, it also helps to relieve the organization's back-office workload, thereby cutting costs.

Self-service applications are ideally web-enabled and provide user-friendly interfaces.

3.2.4 Product Leadership

One way information systems can help to achieve Product Leadership is to support a low time-to-market for products or services. This does not just mean being the first to introduce a new product, but especially catching up with competitors (who enjoy the luxury of a head start). Likewise, short time-to-market is not only important for new products or services, but also for implementing changes to existing ones.

Intuitively, information systems that enable fast implementations of new functionality or modifications to existing functionality can best support a low time-to-market strategy.

3.2.5 Financial performance

Whatever the environment or corporate strategy, reducing costs is always a top priority, especially for commercial organizations. This is not only true for organizations that adopt a 'Cost Leadership' strategy and so require lowest possible cost levels across the organization. All other organizations benefit from lowering costs to increase profit and shareholder value, regardless of their business strategies.

Studies report 60 - 90% of corporate IT budgets are spent on maintaining legacy applications [Bennett 1999], so more efficient maintenance is a prime opportunity for cost reduction in IT budgets. Other opportunities are lower costs for development of new systems and user training. Of course, organizations can not only decrease costs on the IT budget itself, but the right IT can also help to cut costs in other areas, for example through automation.

Some of the approaches mentioned above, like e-Business and self-service customer applications, also cut costs, although this is more a side-effect or secondary goal and not the primary reason to adopt these approaches. Still, they contribute indirectly to cost reductions.

As first mentioned in section 3.2.1, Henderson and Venkatraman state that strategic fit is critical for financial performance.

3.3 Advantages inherent in SOA

This section discusses which features inherent in SOA make it a very desirable architecture for modern organizations by showing how it supports the requirements discussed in the previous section.

3.3.1 Strategic alignment

The strategic alignment model introduced in section 3.2.1 acknowledges systemic competencies as an important area of IT strategy. As example competencies, Henderson and Venkatraman list “system reliability, cost-performance levels, interconnectivity [and] flexibility”, each of which is promised to be well supported by SOA. The next sections describe in more detail how these attributes are supported by SOA.

3.3.2 Integration and standardization

As discussed in the previous section, integration between applications is important for nearly every business strategy. Most organizations have to deal with a collection of various information systems, which can reside inside or outside of the organization. Reliable and flexible integration between these systems is indispensable to create competitive advantages, high levels of process automation and high levels of customer service, among other advantages.

The heterogeneity of these systems has historically made integration complex and expensive. SOA is built on a wide range of open standards have been designed with flexibility in mind. IT vendors acknowledge that integration has been a major problem with previous products because of the lack of flexibility and standardization, and now are widely supporting and developing these new standards.

Whether true out-of-the-box interoperability will be achieved remains to be seen, as vendors have historically deviated from standards to differentiate themselves from competitors. On the other hand, customers are now demanding standards compliance because of the problems that arise from heterogeneity in information systems.

3.3.3 Flexible architecture

Flexibility of information systems is another desired attribute that serves a number of business strategies. The most important one is business agility, the ability to change fast in order to meet changing demands. According to Gartner analysts Roy Schulte, IT systems have historically been “built to last” instead of being “built to change”.

SOA aims to address this problem by allowing systems to be componentized into independent pieces (called services), which can be linked together easily and dynamically to create information systems that support changing business processes. “[S]ervice-oriented systems are becoming the de-facto approach to bridging the gap between business models and software infrastructure and flexibly supporting changing business needs” [Kontogiannis 2007]

3.3.4 Low costs

Cost reduction is one of the key selling points for SOA technology. Not surprisingly, market research by has shown cost reduction is currently the number one business driver for SOA adoption [West 2006]. SOA promises to simplify legacy system maintenance, which currently drains large parts of organizations' IT budgets (see section 3.2.5). According to an industry white paper by webMethods, “reduced skills and effort to support business change [and] price/performance optimization based on freedom to select platform, technology, and location independently” [webMethods 2005] are two more ways SOA can reduce costs directly.

Vendors also claim that SOA can increase developers' productivity by stimulating reuse of code and simplifying integration.

3.3.5 Business process support

As discussed in section 3.2.2, Ward and Peppard state that, to determine which role IT can play in BPR and BPM, one first has to answer two questions first. This section explores the answers to both of these questions.

Support for business process transformation

The first question is “how can business processes be transformed using IT (based on a full understanding of the capabilities of IT)?” [Ward 2003].

Business processes can be modeled using specialized software tools to design, document and even benchmark these processes. These tools are usually used by business users and focus on the business process, with little or no regard for IT support. One common notation (or language) for these models is the Business Process Modeling Notation (BPMN, see [OMG 2006]) (see figure 3.4 for an example model).

One of the technologies usually employed in SOAs are BPEL engines (BPEL stands for Business Process Execution Language). These allow detailed models of business processes to be directly and

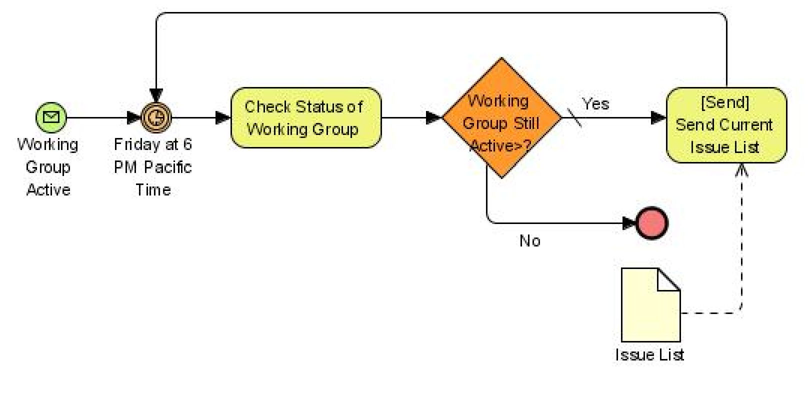
Working Group Active
Friday at 6 PM Pacific Time
Check Status of Working Group
Working Group Still Active>?
Yes
No
[Send] Send Current Issue List
Issue List

Figure 3.4: Example BPMN model (source: [WP 07a])

automatically executed. Since BPEL and BPMN are similar, BPEL models can be based on BPMN (or other) models of business processes. By extending the BPEL models to specify in detail which software services to invoke, the abstract business model can now be executed.

The advantage to this approach is that business users who design processes and IT developers can now work together to design, implement and maintain business processes and the supporting IT systems. A secondary, but nonetheless very important, effect of using BPEL is that IT systems' behavior and workflows are now modeled explicitly and centrally, instead of being embedded in and distributed over the applications' code or even implicit in users' memories (or both). This means processes and applications are transparent and far easier to change.

BPEL engines can also support implementing changes to processes on-the-fly. This means that running instances will keep running according to the old version, while new instances use the newer version. This allows for processes to be changed without halting the system and the resulting down-time.

Another instrument which is supported by SOA is Business Activity Monitoring, or BAM. It is defined as “provid[ing] real-time access to critical business performance indicators to improve the speed and effectiveness of business operations.” [Gartner 2002] BAM can be used as an analytical tool or benchmark for business processes to find bottlenecks or other opportunities for improvement. BAM and its “relative” Business Intelligence (BI) are not exclusive to SOA, but typical SOA architectures and middleware (such as BPEL engines) make it possible to use BAM with little or no required modifications to applications, since their activities can be monitored “from the outside” by tracking BPEL instances and other means, without requiring the application to report its activities “by itself”.

Support for business processes

The second question posed by Ward and Peppard is “how can IT support business processes?”. Next to the possibilities almost every IT system can offer in support of business processes – some of which have already been explored in this chapter – only those that are unique to SOA are discussed here.

Apart from the advantages described above, BPEL engines also allow users to monitor the progress of running process instances and view information about closed or aborted instances. This information can be used to reveal the status of particular instances and help when investigating problems. Together with the information provided by BAM tools, SOA offer more control on and more information about business processes, both of which can help organizations improve said processes. One additional advantage is that this information also can also help organizations monitor service level agreements (SLAs), both those of external services as well as those provided by the organization itself.

3.4 Advantages compared to Oracle Forms

Aside from the advantages inherent to SOA described in the previous section, Oracle's SOA solutions offer some additional advantages when compared to their Forms framework, which is discussed in detail in the next chapter. These differences provide additional reasons for customers to switch to an SOA.

3.4.1 Application user interface usability

User interfaces for Oracle Forms are designed primarily for "power users" who often perform the same activities. The usability is geared towards efficiency, not user-friendliness, and requires users to be trained before they can use interfaces at all. This is especially problematic for applications intended for users with little or no experience with Oracle Forms, like external business partners or end customers.

User-friendly interfaces can be developed using a large number of technologies that are not tied to SOA, or any enterprise IT architecture for that matter. Customers that wish to adopt Oracle's modern application development framework (based on Java) need not automatically adopt SOA as enterprise architecture.

Likewise, SOA does not restrict the choice of user interface technology. In fact, its architecture even promotes the use of different presentation techniques for the same business logic by separating the presentation from the business logic (see the previous chapter for more details). This means that while SOA adoption is not required for organizations that wish to migrate to modern development frameworks, it does offer some advantages. For example, SOA can be used to "connect" a modern user interface developed in a new technology (Java) to existing businesses logic developed using older technology (Forms). This enables a gradual migration between old and new development frameworks and technologies. More examples for the usefulness of SOA with regard to "old" user interfaces are discussed in chapter 2.

3.4.2 Integration

Oracle Forms applications offer no interface to other applications. The only means for integration is to access the database directly. Apart from the implicit standardization issues (as discussed in section 2.4.3), this approach makes it problematic or downright infeasible to integrate external systems, since direct access to the database is undesirable from both technical and organizational perspectives.

As has been discussed in section 3.3.2, SOA provides an ideal environment for application integration, based on open standards and a focus on integration problems.

3.5 Summary of advantages

The primary goal of this chapter is to compile a list of features that enable SOA to better support business drivers than legacy systems, and especially those built on Oracle Forms technology. This list is used as input for the method described in chapter 6, and is based on the major advantages of SOA presented in the previous sections. To make the list usable and comprehensive, a high level of abstraction has been chosen.

The list consists of the following features that make SOA adoption desirable when considering common business drivers:

- **Cost reduction**. As has been shown in section 3.3.4, SOA adoption can result in improved financial performance. Market research shows that cost reduction is currently the main driver for SOA adoption.
- **Integration support**. One of the cornerstones of SOA is the focus on information system integration aspects. The use of open interfacing standards and technologies aimed specifically at supporting integration (like UDDI and BPEL) make SOA the ideal platform for Enterprise Application Integration (EAI) and integration with external entities (e-Business, especially B2B).
- **Business process support**. Section 3.3.5 discussed how SOA supports an organization's BPM and BPR activities and offers a flexible IT architecture that can change along with business processes. This flexibility creates business agility, which is required to gain and maintain competitive advantages. Furthermore, BPEL and BAM software provide valuable (real-time) information about running business processes which organizations can use to monitor and benchmark these processes.
- **User-friendly applications**. User-friendly applications are not a property inherent to SOA. More user-friendly interfaces can be developed using any one of a large set of technologies and frameworks currently available. However, by providing a platform that enables gradual migration and reuse of existing logic, SOA is an suitable enterprise architecture for organization that wish to leave the Oracle Forms framework in favor of a more modern framework.

It is important to note that these four high-level features of SOA are not entirely independent of another. For example, SOA's focus on integration can lead to a decrease in maintenance effort for applications that need to be integrated. This leads to increased business agility (because IT can adapt faster) as well as to cost reductions (since developers need less time for maintenance). Figure 3.5 shows the dependencies.

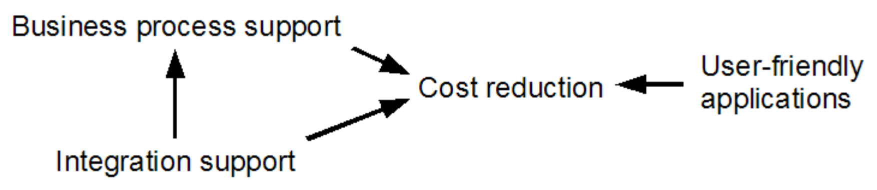

Figure 3.5: Dependencies between advantages of SOA

Chapter 4: Leveraging Approaches

The third chapter has shown how SOA is better suited to support current business drivers like business agility and improved financial performance. The discussion of the differences between legacy architectures and SOA in chapter 2 should make clear that it is not possible to simply transform or "migrate" legacy applications in order to get a system based on SOA. By definition (see section 2.1.1), legacy applications cannot be abandoned or redesigned from scratch because of the huge investments that have been made over the years.

But is it really necessary to make every application completely SOA-based? Sure, SOA offers a number of advantages over legacy architectures. (One of these, cost reductions, is always beneficial, regardless of business strategy or environment). However, since complete transformation is generally infeasible, is it perhaps possible to partially transform legacy applications in a way that makes them fulfill business requirements? This chapter investigates these possibilities.

The first three sections provide a theoretical background based on the literature on the subject. The last three sections discuss actual approaches to leverage or replace legacy applications.

4.1 Introduction

The first chapter introduced the notion of "leveraging" legacy applications. This section will discuss in detail what is meant by that.

4.1.1 Definitions

Before we start discussing modernization techniques and approaches, we should first define the terminology that is (mis)used in the descriptions of the approaches discussed in the next section. [Chikofsky] presents a taxonomy of terms (in the context of reverse engineering and design recovery) that are used throughout the literature.

In this thesis, a important distinction is made between the terms “maintenance” (as defined below) and “leverage”. The difference is that the latter aims to add value to a system, for example by adding new functionality or improving its flexibility or reducing maintenance cost.

Software maintenance *“is the modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a changed environment” (according to ANSI/IEEE Std. 729-1983)*

Reverse engineering *is the process of analyzing a subject system to*

- *identify the system's components and their interrelationships and*
- *create representations of the system in another form or at a higher level of abstraction*

Reverse engineering *does* not *involve changing the subject system or creating a new system based on the reverse-engineered subject system. It is a process of* examination*, not a process of change or replication.*

Restructuring *is the transformation from one representation form to another at the same relative abstraction level, while preserving the subject system's external behavior (functionality and semantics).*

Re-engineering *is the examination and alternation of a subject system to reconstitute it in a new form and the subsequent implementation of the new form. It generally includes some form of reverse engineering followed by some form of forward engineering or restructuring.*

4.2 Classification of modernization approaches

[Papazoglou 2006] presents an overview of legacy modernization strategies, which can be divided into invasive and non-invasive approaches. The discussion is presented in the context of e-Business integration.

The authors begin by stating that applications consist of two conceptual parts: their environmental part and their business part. The first manages the application's environment such as hardware, operating system, communication infrastructure and database. The second “deals with its perceived business functionality [and] contains the application's businesses rules and businesses process flows.” In the context of this research, the environmental part does not need to be considered, since legacy applications run on the same environment as SOA solutions.

Non-invasive approaches

Non-invasive approaches keep the existing application flow intact and result in new presentation interfaces for the legacy system by employing screen-scraping or using the systems APIs or new wrappers to access existing business functionality. Papazoglou et al. present three alternatives for this approach: “refacing“, “repurposing“ and “presentation tier modernization techniques“.

Unfortunately, Papazoglou et al. do not provide detailed definitions or descriptions, making it hard to tell the exact differences between the categories. What the approaches have in common is that they do not change the legacy system, but only substitute or modernize the user interface in order to make them fit into modern desktop environments or make them more user-friendly.

Invasive approaches

Invasive approaches actually modify the legacy system, which makes these approaches inherently more powerful and flexible than non-invasive approaches. However, it also makes them costlier and riskier, since production systems are modified, which can introduce errors. Papazoglou et al. discern three alternatives:

- *Maintenance*, which is characterized by iterative and incremental changes to correct small deficiencies or add minor enhancements.
- *Replacement* of the whole or parts of the legacy system by an an off-the-shelf product or a newly developed custom system. This alternative is usually chosen when the legacy system can not longer be maintained efficiently or when business needs change so much that the costs of making the required changes are higher than the economic value of the system.
- *Re-engineering and transformation* (also called *modernization*). This entails analyzing the legacy system to understand what it does and how it works, in order to modify some parts of even redevelop them using newer technologies. The first step is very important since the legacy system already partially satisfies business requirements.

The authors state that the use of the first two techniques is on the decline. Maintenance of large legacy systems does not significantly improve their value, nor does it create maintainable assets. Replacement is often infeasible because it means abandoning the large investments that have been made (see the introduction to chapter 1 for more details). This leaves the last option as the most viable alternative.

Modernization approaches

According to Papazoglou et al., re-engineering can be divided into two types: white-box and black-box re-engineering.

- “White-box re-engineering concentrates on reverse engineering techniques to gain an understand of the internal structure and operations of an application.” [Papazoglou 2006] After the original program is sufficiently understood, it can be restructured at the application and code level to decompose the system into self-contained pieces of code, or components.

- Black-box re-engineering aims to re-interface existing code (often based on wrapping) by analyzing the inputs and outputs of the system instead of its source code. “In general, the black-box approach is often preferred to white-box re-engineering because the technology for interfacing and integration is developing much faster than the technology for program analysis and understanding.” [Papazoglou 2006] On the contrary, Jha and Maheshwari ([Jha

2005]) argue that black box wrapping techniques are impractical because they usually require white box techniques to be applied in order to sufficiently understand the legacy system, so one might just as well go for white box approaches in conjunction with reverse engineering.

By definition, re-engineering implies restructuring an existing asset, which can cost considerable effort. Another modernization approach is legacy componentization, which aims at breaking up a legacy system into isolated parts (called components) with well-defined interfaces.

"[L]egacy componentization involves surrounding the legacy system with a layer of component software that hides the unwanted complexity of the old generation system and reveals a modern interface to give it a new and improved functionality." [Papazoglou 2006] This abstraction makes it possible to use components using modern technology and programming environments. It also allows for the legacy component to be replaced by a modern component without changing the the rest of the system. This makes it possible to gradually replace legacy technology with modern technology.

4.2.1 Evaluation of approaches

Non-invasive

Non-invasive approaches all apply mainly to modernizing interfaces based on mainframes and terminals. This does not apply to Oracle Forms interfaces, which are already graphical in nature and run on most modern desktop environments (and even on the web). However, since user-friendliness is one of the requirements where Forms applications (may) need improvement, presentation tier modernization techniques must be considered.

Papazoglou et al. also mostly use mainframe systems with terminal screen interfaces running on legacy hardware as typical examples of legacy systems. This assumption does not hold for Oracle Forms applications. As section 2.1.1 makes clear, by the definition used in this thesis, "legacy" systems are not only those that run on unsupported hard- and software, but any systems that resist change. Of course, the former type of system usually resists change automatically, for example because skilled developers are scarce and hence expensive.

Oracle Forms applications are not legacy because the technology is outdated or unsupported (it isn't). They are considered legacy because their architecture and technology resist change. This implies that these applications do not necessarily need to be "migrated" to newer development frameworks or newer hardware environments, but instead might be improved through non-invasive techniques. This is another reason to consider non-invasive techniques.

Invasive

Since maintenance on the lowest level of abstraction does not improve the assets value for the long-term and replacement is often not a viable alternative because of the huge investments made in legacy assets, re-engineering and transformation are left as "the most appropriate approach to legacy system integration." Cormella-Dorda et al., in a survey of legacy system modernization approaches, elaborate on this conclusion by stating that interface modernization techniques like screen scraping (which, according to the authors, are derogatorily called 'whipped cream on road kill') do not improve the system's value, since, “from the IT department's perspective, the new system is as inflexible and difficult to maintain as the legacy system.” [Cormella-Dorda 2000]

Papazoglou et al.'s conclusion about re-engineering and transformation as the most appropriate options is drawn in the context of modernizing legacy systems for the purpose of e-Business integration. We have to be careful not to generalize this conclusion to the context of leveraging legacy systems in SOAs. Since integration is but one way to achieve this, we cannot disregard non-invasive approaches, especially since they are often easier and cheaper to apply than invasive approaches.

4.3 Decisional framework

De Lucia, Fasolino and Pompella [de Lucia 2001] present a decision framework, which distinguishes several alternatives for the management of IT assets (based on [Verdugo 1988]), being “ordinary maintenance, reverse engineering, restructuring, re-engineering, migration, wrapping, replacement with COTS [common off-the-shelf system], and discarding”. To select one of these approaches, two factors have to be considered first: the system's 'business value' and its 'technical value', each of which is ranked on a two-score ranking scale, as shown in figure 4.1.

In the context of this research, the business value of the legacy systems is high because they support essential business processes and because businesses have already invested heavily in these systems.

Technical value			
	I	II	
High	Evolution / Massive Adaptive Maintenance	Ordinary Maintenance	
	III	IV	
Low	Elimination / Replacement	Reverse Engineering / Restructuring / Transformation	
	Low	*High*	*Business value*

Figure 4.1: Quadrant map for portfolio analysis (adapted from [de Lucia 2001])

Ranking the technical value on a binary scale is not as straight-forward as it might seem. The monolithic architecture of Oracle Forms systems and the inherent tight coupling justify a “low” ranking (which is why they are often considered “legacy” in the first place). However, the hardware and software environment is still being actively developed and supported, and user-friendliness, performance and scalability certainly do not pose any major problems or limitations. Surely a “high” ranking is in order if these qualities are more important than loose coupling.

We can conclude the technical value depends on the reason to evolve the legacy system, and has to be rated on an individual basis. For example, if integration and reuse of code are important reasons to evolve the system, the technical value should be rated “low” because the monolithic architecture significantly resists these goals. This is exactly the context in which Papazoglou et al. recommend invasive approaches, which corresponds to the approaches shown in the quadrant labeled “IV” in the figure above.

4.3.1 Evaluation

To elaborate on the discussion in section 4.2, we can also categorize the approaches described above by looking at the characteristics of the resulting system (instead of the characteristics of the approach).

The first category of approaches do not fundamentally change the system's architecture and/or base technology, and hence do not address their root problems, making them short-term solutions ([Papazoglou 2006], [Jha 2005], [Zhang 2004]). To put it another way, they do not address the problems that make the system to be considered “legacy” in the first place. As [Bianchi 2003] puts it, “if the wrapped system needs to be evolved in some way, all the consequences of the aging symptoms indicated in [Vis97] will re-emerge. Therefore, [...] the wrapping approach [...] does nothing to solve the problem of maintenance of aged programs.”

The second category, which includes re-engineering, and discarding, does aim to address the underlying root problems and results in systems that are not considered legacy anymore.

Another way to put this is to say that the first approaches address (one or more) *symptoms* of legacy systems, while the latter address the root causes for these symptoms. If we think about it this way, we can justify calling the first category short-term solutions, while the latter are long-term solutions.

Not surprisingly, we must conclude that the long-term solutions are always much costlier and harder to implement, or even downright unrealistic, as is usually the case for the “discarding” approach. In practice, however, given that the alternative long-term solutions are also very costly (“[i]n general, the advocates of reengineering tend to underestimate the difficulty of reusing legacy code” [Sneed 2001]), discarding should always be considered.

Jha and Maheswari have conducted a literature study and state that “there is a lack of literature on successful modernization processes. Many modernization projects fail as outlined by the Standish Group.” [Jha 2005] They go on to summarize that most organizations consider redevelopment approaches risky, and that “the reverse engineering of procedural components of a large application is still unsolved.”

The thing to take away from this discussion is that short-term solutions should be preferred to long-term solutions in the method, because the former will deliver the business requirement faster, and with less effort and risk.

4.4 Leveraging Oracle Forms

The preceding section has presented general approaches and techniques that can be employed to modernize legacy applications or transform (re-engineer) them into a new form in order to mitigate (or eliminate) their shortcomings. As has been discussed in section 3.1, the goal should not be to dogmatically adopt the latest technology or migrate to the latest environment, but rather to change the system in a way that makes it fulfill business requirements.

A literature survey of detailed modernization approaches did not result in any descriptions that were relevant to the particular problem covered in this thesis. A lot of papers deal with a fundamentally different kind of legacy systems, e.g. written in COBOL or C and/or running on mainframe systems. Unfortunately, the assumptions these papers make about legacy systems just do not hold for Oracle Forms. Other papers describe very abstract techniques which cannot be readily applied and which also have not been tested. In fact, none of the detailed techniques seem to have been proven to work on industrial scale software or large projects [Bennett 1999].

The following section presents approaches and techniques that enhance the value of Forms applications or leverage these in an SOA. The knowledge is largely based on interviews with Oracle Forms experts working at the Dutch office in De Meern.

4.4.1 Cost reduction

The discussion in section 3.3.4 presented multiple ways in which SOA can reduce IT costs. Since the technology underlying Oracle Forms is proprietary, customers do not have freedom of choice, which eliminates one factor that can lead to decreases in costs (according to [webMethods 2005], see section 3.3.4). It also means specialized developers are required to develop and maintain applications. This problem will worsen with time as more and more Forms developers (in-house as well as Oracle's and its partners') switch to modern development environments ([Gartner 2007]).

Unfortunately, Forms is fundamentally monolithic in nature, which is another reason maintenance will always remain costly. Following Oracle's advice of keeping clients thin and centralizing logic in the database can help to make applications more manageable, but even in optimal circumstances, application logic is still tightly coupled with presentation logic.

It is possible however to lower costs by increasing developers' productivity through reuse of code and simplifying integration. Both aspects can be partly supported by Forms – see the next section for details.

Another way to lower costs is by making applications more user-friendly so end-users need less training (or none at all). See section 4.4.4 for more details.

4.4.2 Integration

Zhang and Yang have done research on the topic of extracting logic from legacy systems and describe “a component-based, service-oriented approach to recover services in legacy systems.” [Zhang 2004] In the introduction to their paper, they distinguish two ways to leverage legacy systems in an SOA, namely by making it either consume or provide services. The authors conclude that the latter is more important to investigate because it offers more possibilities. This is certainly true for this research, although we cannot disregard alternatives that employ the first approach, since it will almost certainly always be the easier one.

Consuming services

Consuming (web) services from Oracle Forms is already supported “out of the box”[4]. Recent versions of Forms have been integrated with Java, which allows Forms modules to call PL/SQL functions which act as wrappers for Java classes. These Java classes in turn can be proxies that call services (which in turn can be used to start BPEL process instances) and return the result, if any.

Providing services through componentization and wrapping

Oracle Forms can not provide services to other applications. It is possible however to wrap stored procedures and PL/SQL functions with some Java code to make them callable as (web) services.

However, this requires the logic to be implemented in the database as callable functions or stored procedures. Unfortunately, application logic in Oracle Forms can also be implemented in the client application and may even be distributed among more than one module (see appendix A). While Oracle has advised developers to centralize logic in the database and keep client applications as “thin” as possible, developers have disregarded this advice.

To enable reuse of logic through callable services, the logic must first be moved from the application tier to the database. This can be relatively easy when the logic is already isolated in PL/SQL functions (in the client application), in which case they can be easily moved to the database, and applications only need to be updated to call the same function at a different location.

However, when logic is not isolated but instead spread out over one or more modules, isolating and moving the logic can take a lot of work and require a lot of modifications and testing.

Even in the cases where logic is already centralized in the database, the functionality of the resulting services is predetermined bottom-up instead of in a top-down manner. This means that instead of investigating which (coarse-grained) services are sensible or required and implementing these, (fine-grained) services are built based on what functionality is readily available. This might not always be desired in order to reuse services in the same or other applications (which is the whole point of building services).

Fortunately, fine-grained bottom-up services *can* be used to create coarse-grained services by either building composite services based on a few low-level services or by enhancing low-level services in their Java implementation, for example by running a few extra queries or performing some transformations, etc.

We can conclude that it *is* possible to create services by reusing logic built into Forms applications, although these will likely be fine-grained and might require some initial effort to recover and isolate the original PL/SQL logic and move it to the database tier.

4.4.3 Business process support

One of the major problems with regard to business processes and Forms applications is that processes are modeled implicitly in the application's code (see section 2.4.2). For example, the application might present different interfaces or enforce different data constraints, depending on

4 See http://www.oracle.com/technology/products/forms/pdf/10gR2/forms-soa-wp.pdf

some value (e.g. a customer's membership status). This not only makes it difficult to change these processes, it also means that just understanding what the current process *is* requires white-box reverse engineering.

The situation is often even worse, when business processes are not even manifested in (and hence enforced by) the application at all, but instead are only in the minds of one or more employees. In this case, the application itself does not guide (or restrict) the user at all to make sure policies and processes are enforced, but it is left up to the user to open the right screens in the proper order and enter the necessary information. One needs only consider what happens to running processes when the only employee who knows the process is on leave or sick (or leaves the company) to see why this is problematic.

As described in section 3.3.5, SOA enables business process modeling and allows real-time monitoring of processes through BPM, BPEL and BAM. Fortunately, there are a few alternative solutions to tie Forms applications to BPEL and BAM engines.

Human workflow

The first is to employ Business Process Modeling to explicitly model current business processes. These can be recovered by investigating application code and / or interviewing employees. Alternatively, a new or revised process can also be designed from scratch in the case of BPR projects. Of course, simply modeling business processes is always possible and provides only limited benefits. Executing process models in BPEL engines delivers far greater benefits.

This can be achieved by using the “human workflow” functionality of BPEL engines to create human task lists. By opening a “task” from the list, the user is taken to the corresponding screen in the Forms application. Additionally, more information on the task can be displayed, and certain input fields can be filled in by the application.

In this situation, the business process is modeled explicitly in BPEL (instead of implicitly in the application or even not at all). The BPEL engine now guides users through the Forms application.

Depending on the design of the screens and the extent to which business process rules are already hard-coded into the application, this approach can take from a little to a lot of effort to change the Forms application to make it usable in this way.

Interfacing with BPEL

There are a few more alternatives to interface Forms applications with BPEL engines. For example, it is possible to start BPEL processes by invoking a service, as discussed in the previous section. Instead of using the BPEL engine's web service interface to communicate, the Java runtime in Forms can also invoke the BPEL Java API directly[5].

The BPEL engine can also invoke a service which modifies data in the database in some way which is useful to a Forms application, for example to set the value of some number of fields to desired values.

5 See http://www.oracle.com/technology/oramag/oracle/05-mar/o25forms.html

We can conclude that communication in both directions is possible, which enables a large number of custom applications of this approach.

Business Activity Monitoring

BAM engines usually use information about BPEL process instances to present overviews and allow monitoring. Alternatively, it is possible to send messages through into an Enterprise Service Bus (ESB) to publish "events", which can be monitored by a BAM engine. One example application is to send an event message every time the "Save" button of a screen for new customer orders is hit to monitor the number of orders.

4.4.4 User-friendly applications

As has been discussed in section 3.4.1, Oracle Forms user interfaces are geared towards "power users" and are not particularly user-friendly. This is a problem if an application is supposed to be used by business partners or end customers, as they can't be expected to be familiar with Oracle Forms interfaces. In these cases, a new user interface has to be developed.

Unfortunately, since the presentation logic is tightly coupled with the rest of the logic in Oracle Forms, it is not possible to develop a new interface which makes use of the logic implemented in Forms modules. Fortunately, the componentization approach described in section 4.4.2 enables limited reuse of code.

A different alternative has been developed by one of Oracle's customers. This technique allows to integrate Forms applications (including their interfaces) to be seamlessly integrated[6] in Oracle's Java development environment, ADF. This allows developers to create modern interfaces using a J2EE environment but integrate (parts of) Forms applications into these interfaces. Communication between the environments is possible in both ways. This technique enables reuse of existing Forms applications in modernized user interfaces and allows gradual migration towards a modern, Java-based development environment.

4.5 Replacing Oracle Forms

The previous section presented some invasive and non-invasive approaches to leverage legacy applications built on Oracle Forms technology in an SOA. While these do provide a number of opportunities to leverage Forms applications in an SOA and mitigate the problems common to "legacy" technology, none of the approaches result in a modern, easy to change and flexible information system. Since the Oracle will only make small investments in Forms, and support for the platform will cease eventually, Forms is on its way to become completely obsolete.

This is why, for the long term, Gartner recommends organizations to "[a]pproach Oracle Forms within a 'containment' strategy [and] migrate to industry-dominant technologies [...] to align with industry best practices in the future." [Gartner 2007] In other words, applications developed with Oracle Forms will have to be replaced by more modern applications sooner or later (Gartner recommends organizations to migrate within five to ten years). This means that replacing the legacy

6 See http://www.oracle.com/technology/pub/articles/wilfred-adf-forms.html

application should always be considered in parallel to any of the leveraging approaches discussed in the previous section.

This section introduces approaches that lead to long-term solutions by replacing the legacy system altogether. These approaches can be divided into two major types. The first, called “Big Bang” or “Cold Turkey”, entails buying or developing a new application from scratch and chose a date at which to switch from the old application to the new application. This is a very risky approach and is often infeasible [Bisbal 1999], which is why it won't be considered here. The second type uses an incremental or gradual migration from the old application to the new. Only this type will be considered in this section.

4.5.1 Purchase of standard software

The first option for replacing a legacy system is buying a new COTS (commercial off-the-shelf) application. This alternative is usually cheaper and faster to implement than developing a new system from scratch. However, the system may have to be thoroughly customized to fit the organization and its business practices, which negates the advantages of buying a COTS.

The alternative is to change the organization to fit the application through business process redesign (BPR). Because of its radical nature and the risk involved, BPR is an infamous approach known for its high failure rate [Papazoglou 2006]. However, there may be cases where business processes are outdated and / or wasteful and are only held in place because the legacy system cannot be changed. In these situations, the replacement of the system might be a good opportunity to modernize and optimize business processes as well.

This approach does little to leverage the investments made in legacy code, although it is still possible to gather the requirements for the new system by studying and possibly reverse-engineering the legacy system.

4.5.2 Development from scratch

The alternative to buying (and customizing) a new system is to develop one from scratch. The result is a custom-built system which fits the organization. The downsides are cost and time compared to buying a COTS system, and the cost of maintaining a custom application.

The remarks about BPR made in the previous section apply to this approach as well. If the new system is developed to be an equivalent of the legacy system, bad business processes will remain. The development of a new system is a good opportunity to change business processes, but care should be taken to avoid radical changes. Depending on the situation, copying the legacy system's functionality and changing it along with business processes afterwards might be a better alternative.

With regard to leveraging the value of the legacy system, redevelopment is better suited to reuse legacy code. Legacy functionality implemented in the database can be componentized and wrapped (see section 4.4.2), and can be reused in the new system. The literature features several frameworks and methodologies for leveraging and re-engineering legacy systems. Two of these are the Options Analysis for Re-engineering (OAR) [Bergey 2001] and the Service-Oriented Migration and Reuse Technique (SMART) [Lewis 2005] (see section 4.6).

Development from scratch is also better suited for incremental replacement. Although a COTS system can naturally be bought in one piece, customized and then taken into use gradually, the costs of buying the system have to be made at the beginning of the project. Additionally, any change in requirements later in the replacement process are costlier because the COTS has already been paid for. This is opposed to a redevelopment approach, where the cost of development is spread out over time, as are the decisions about which requirements the system must fulfill.

4.5.3 Migration

The previous two sections have already touched on the subject of migration from the old system to the new. Bianchi et al. present three approaches, all similar to the original "Chicken Little Strategy" described by Brodie and Stonebraker in 1995. This approach entails "gradually rebuild[ing] the legacy system on the target platform using modern tools and technology. During migration, the legacy and target system form a composite information system [...]" [Bianchi 2003]

The three approaches presented by Bianchi et al. differ on how to deal with the legacy database. The reason for the difference in handling databases is the assumption that the legacy database cannot be used for the new system. As has been shown in appendix B, the data tier used by Oracle Forms applications is the Oracle database, which is not a "legacy" database, but instead is used for modern applications as well. This means it is not necessary to migrate, duplicate and re-engineer the data tier. Since the database does not require any special treatment, migrating from a Forms application to a new application only involves the application tier.

We can conclude that the most suitable way of migration is to develop the new system on its target platform, and use it in parallel with the legacy. New functionality can be developed and taken into use in a step-by-step fashion, until the legacy system is completely replaced.

4.6 Existing methodologies

The literature contains a large number of methods for migrating or re-engineering legacy systems. However, many consider complete migration the main goal, without evaluating which parts need to be migrated or considering the business benefits of the migration. In short, the goal of many methods is to migrate the legacy system to newer technologies and architectures, instead of adopting a more pragmatic short-term view and consider which aspects or parts of a legacy system resists business drivers, and find a lightweight solution to fulfill these drivers.

This section presents a brief overview of work related to the method presented in this thesis. Some of the elements of the methods developed by other researchers have been used as a basis or inspiration for the method presented in chapter 6.

4.6.1 Options Analysis for Re-engineering (OAR)

Bergey et al. have developed a method that can be used to identify and “mine” reusable parts of legacy systems that can be used for organizations that wish to implement a new software system (either COTS or developed from scratch). The authors describe “Options Analysis for Reengineering (OAR) [as] a systematic, architecture-centric, decision making method for identifying and mining software components within large, complex software systems. Mining involves rehabilitating parts of an old system for reuse. OAR identifies potentially relevant architectural components and analyzes the changes required to use them in a software product line or new software architecture. In essence, OAR provides a set of mining options along with estimates of the cost, effort, and risks associated with those options.” [Bergey 2001] An overview of the method's activities is presented in figure 4.2.

The context of the OAR method is the development of a new system or product line, where legacy system components can be re-used. This thesis acknowledges that re-engineering Oracle Forms applications is the only viable long-term solution (see section 4.5), but focuses on less invasive approaches to leverage these applications in the short to medium term. The techniques presented by Bergey et al. to identify and evaluate legacy components for reuse are applicable to some approaches presented in this thesis.

4.6.2 Service-Oriented Migration and Reuse Technique (SMART)

Lewis et al. have developed a technique “that helps organizations analyze legacy systems to determine whether their functionality, or subsets of it, can be reasonably exposed as services in a Service-Oriented Architecture (SOA). Converting legacy components to services allows systems to remain largely unchanged while exposing functionality to a large number of clients through well-defined service interfaces.” [Lewis 2005] The technique is called SMART, and is derived from the OAR method described in the previous section.

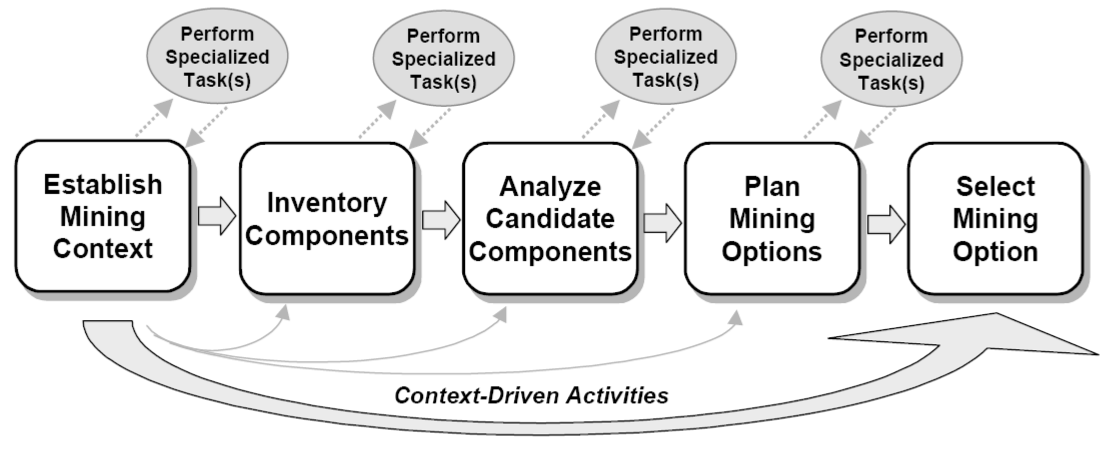
Perform Specialized Task(s)
Perform Specialized Task(s)
Perform Specialized Task(s)
Perform Specialized Task(s)
Establish Mining Context
Inventory Components
Analyze Candidate Components
Plan Mining Options
Select Mining Option
Context-Driven Activities

Figure 4.2: Overview of OAR activities (source: [Bergey 2001])

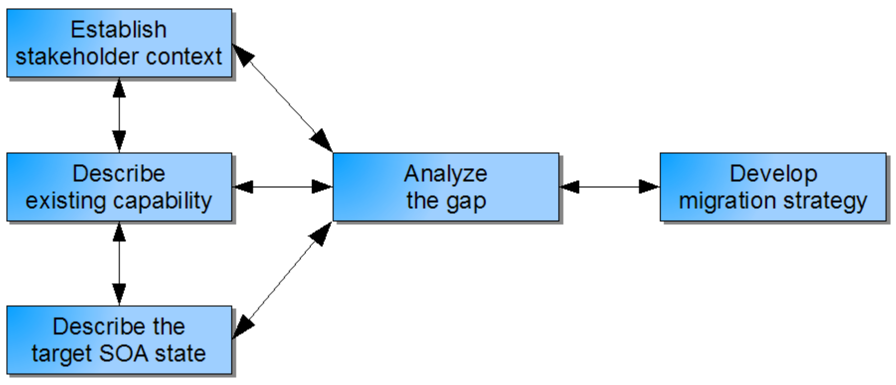
Establish stakeholder context
Describe existing capability
Describe the target SOA state
Analyze the gap
Develop migration strategy

Figure 4.3: The activities of the SMART (source: [Lewis 2005]*)*

The technique consists of five activities (see figure 4.3), starting with three data gathering activities at the beginning of the project. The next step aims to analyze the gap between the current situation and the desired situation. During the last step, a migration strategy is developed. The activities do not have to be carried out sequentially, and iteration is also possible.

Smith, one of the authors, summarizes the activities in one of his papers:

1. **"Establish Migration Context,** which develops an understanding of the goals and expectations of the SOA environment; the programmatic constraints, such as schedule and budget, any previous reuse efforts; and an understanding of the legacy system at a high level. Appropriate stakeholders and candidate services for migration are identified, together with the business processes that they support.
2. **Describe Existing Capability,** which obtains data about existing legacy components, architecture and design paradigms, complexity and coupling, dependencies, change history and historic cost data.
3. **Describe Target SOA State,** which identifies how services would interact with each other and with the target SOA environment, determines the target SOA state, and determines QoS expectations and the execution environment for services.
4. **Analyze the Gap,** which identifies the gap between the existing state and future state, and determines the level of effort and cost needed to convert legacy components into services. In some cases, additional analysis methods such as evaluation of code quality and architecture reconstruction may be needed.
5. **Develop Migration Strategy,** which develops one or more recommended strategies that may include identification of specific components to migrate, recommendations on the ordering of migration efforts, and specific migration paths to follow, such as wrapping vs. rewriting code."

As is the case with the OAR method, SMART assumes a target SOA, and the goal of the method is to find and migrate reusable parts from legacy systems. While this is one of the approaches described in section 4.4, this thesis also considers less invasive approaches as short-term solutions.

This is why neither of the two methods presented so far cover all the options that can be considered when leveraging legacy applications as defined in this thesis. Both methods do however contain elements that are relevant to the method developed for this thesis, which is discussed in section 6.2.

4.6.3 Software As a Business Asset (SABA)

Brooke and Ramage ([Brooke 2001]) wrote a paper about their project, whose goal was "to develop approaches to assist business to make decisions about legacy systems." The authors recognized that legacy systems consist of more than just a technical dimension, and its business environment must be taken into account to understand the system. Instead of developing a rigid set of alternative solutions, they thought it should be possible to "devise ways to help organisations identify the nature of the legacy problem and to assist them in identifying alternative approaches to change."

The result is an interdisciplinary approach, called Software as a business asset (SABA, see figure 4.4), which combines "technical expertise from the software engineering field with theory and method from the organisational development arena." Brooke and Ramage note that very few research teams have worked this way, but instead have mostly focused on either one of the two areas.

SABA is an iterative approach and requires at least one cycle. It starts with the organisational scenarios tool (originally developed by Brooke in 2000), progresses to the technical scenarios tool (TST) and then moves through the OST stage again, until, eventually, a preferred scenario and a suitable solution have been selected. Since the approach is multi-disciplinary, Brooke and Ramage recommend a participant group of about a dozen people, including (taken from [Brooke 2001]):

- Senior directors (preferably including someone at Board level)
- Managers from different organisational functions (including Human Resources)
- IT specialists (preferably including a software engineer)
- Front-line staff (including those at the external customer interface)
- End users (preferably including an external customer)

In the initial stage of the OST, the participants are asked to describe their personal view of the legacy system. After this, the group is asked to develop several scenarios (see [Brooke 2001] for a more detailed description), starting with the status quo, and moving on to scenarios that are increasingly ambitious and radical in terms of the depth of structural change. Each of the scenarios is analyzed against nine criteria (taken from [Brooke 2001]):

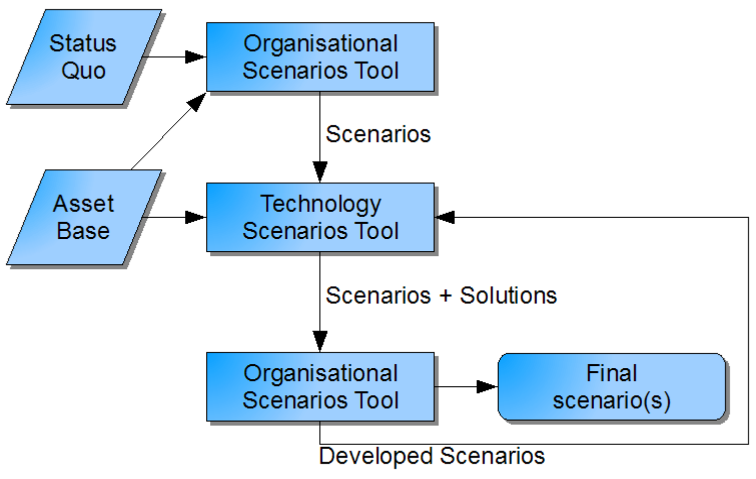

Figure 4.4: Overview of the SABA model (source: [Brooke 2001])

- Boundary: the unit of analysis (e.g. the whole organisation or one area)
- Vision: the overall business approach (e.g. specialised sales)
- Logic: organization rationale for the vision
- Structure: of the organisation
- Roles: organisational roles of people
- View of information: information as an objectified unit of resource (the resource view) versus information as a subjectively interpreted phenomenon (the perceptual view)
- Costs: major costs, both financial and non-financial
- Benefits: both financial and non-financial
- Risks: major sources of risk

At the end of each iteration of the OST, the participants have identified and prioritized several scenarios. The next step is to identify the IT technology required to fulfill each scenario.

In the next stage, the participants should identify the possibilities for technology change for each scenario, resulting in a set of technological options. It is in this phase that existing systems (not all of which need to be "legacy") are considered, as are technical details about them, such as the systems' languages, structures, documentation and software maintenance processes. The possible solutions identified in the first step can be evaluated using the information captured in the second step. Once a preferred solution has been selected, more detailed information about the solution has to be gathered, including (but not limited to) costs, risks, tool investment, expertise.

At this point the business impact of the preferred scenario and its preferred solution should be evaluated by a second iteration of the OST so business experts can consider the strategic implications and the organization's readiness for the required change.

Brooke and Ramage remark that the subjectivity of the SABA approach is both a strength and a challenge. "In particular, the OST is dependent for its exact form on the workshop design and facilitation style of those using it. It is designed to be a tool-in-use, modifiable according to the context." [Brooke 2001] SABA does not always produce the same results, a property which the authors count as a strength because it prepares organizations for change and also reflects the changing business requirements of an organization.

Chapter 5: Requirements

The approaches to leverage and modernize Oracle Forms applications discussed in the previous chapter cannot always be implemented. Every approach poses certain requirements, some of which have already been mentioned in their descriptions. This chapter investigates the requirements that need to be met in order to implement the approaches.

A comprehensive list of requirements would be insensibly large, containing anything from skilled developers, budget, time and more – most of which are either trivial or well documented in both academic and industry literature. Although every approach has its own requirements, interviews with Oracle experts have shown that there are two key areas that play a significant role in assessing the feasibility of any approach: the legacy system's quality and the organization's SOA maturity. This chapter will describe both of these in detail and present a brief discussion of other requirements.

5.1 Legacy system quality

The approaches for leveraging a legacy Oracle Forms application described in the previous chapter pose requirements on the application itself. For example, some approaches require logic to be isolated so it can be reused, while others require a recent version of Oracle Forms in order to call Java code for integration purposes.

This section discusses which aspects of the legacy system might need to be considered to evaluate the feasibility of a particular approach. The set of aspects is based on the non-comprehensive list of approaches in presented section 4.4, and therefore might not be complete. Therefore, it might be necessary to extend the set of aspects. To this end, this section first discusses how a quality model can be used to formalize the set of aspects and define measurements for each.

5.1.1 Quality model

ISO 9126 [ISO/IEC 2001] is a standard for measuring software quality aspects. It defines software quality as “[t]he totality of features and characteristics of a software product that bear on its ability to satisfy stated or implied needs.” [Fenton 1997] According to the standard, quality can be (comprehensively) decomposed into six factors:

- Functionality
- Reliability
- Efficiency
- Usability
- Maintainability
- Portability

Each of these factors can be refined through multiple levels of subcharactersitics. Although the standard itself does not address these, an annex to the standard (ISO 9126-1, see [SQA]) contains examples for (only) the first level of refinement [Fenton 1997]. Furthermore, the QUINT2 model ([SERC 2005]) extends the ISO model with more characteristics and provides

Not all factors and subcharaceristics are relevant as requirements for the leveraging approaches. Although the list of approaches in the previous chapter cannot be considered comprehensive, it is still reasonably possible to identify a comprehensive subset of factors and subcharacteristics that can be relevant to any leveraging approach. However, new leveraging approaches may very well pose requirements on areas that are not covered here, in which case the subset will need to be extended (also see section 8.2). The factors and subcharacteristics considered in this thesis are presented in table 5.1 and form a subset of the ISO 9126-1 and QUINT2 model.

The subcharacteristics presented in table 5.1 can be refined further into measurable attributes. The QUINT2 model defines multiple indicators for each attribute. The ISO 9126-2 and ISO 9126-3 standards (see [ISO/IEC 2001]) describe external an internal metrics, respectively.

Each leveraging approach can pose requirements on the legacy system's quality. The ISO 9126 quality model and its extensions (ISO 9126-1, QUINT2) can be used to refine the requirements and guide the evaluation of the system's compliance to these requirements. Unfortunately, these models are complex, large-scale and abstract, making them unfit to use directly in the context of the problem addressed in this thesis.

For example, the QUINT2 model measures changeability with seven rather abstract indicators (most of which are taken from the original ISO model), which are time-consuming to measure and do not provide detailed measurements which can be used to evaluate the feasibility of a particular leveraging approach.

Factor	Subcharacteristics	Description
Functionality	Interoperability	A given software component or system does not typically function in isolation. This subcharacteristic concerns the ability of a software component to interact with other components or systems.
Maintainability	Analyzability	Characterizes the ability to identify the root cause of a failure within the software.
	Changeability	Characterizes the amount of effort to change a system.
	Stability	Characterizes the sensitivity to change of a given system that is the negative impact that may be caused by system changes.
	Testability	Characterizes the effort needed to verify (test) a system change.
	Reusability	Characterizes the potential for complete or partial reuse in another software product.

Table 5.1: Relevant factors of the ISO 9126-1 and QUINT2 models (sources: [SQA], [SERC 2005])

To allow quick (perhaps even partly automated) evaluation of a system's compliance to requirements posed by a particular leveraging approach, a more specialized quality model could be developed, based on the ISO and QUINT models. For example, when measuring changeability, a metric could be developed that takes into account particular aspects of Oracle Forms applications which are known to affect changeability, like the fraction of PL/SQL code isolated in reusable procedures (as opposed to embedded into the main program code – see section 4.4.2).

Another benefit of developing specialized metrics is that it should be possible to write software that analyzes a system's source code to either assist experts in measuring or even provide measurements automatically. This would enable fast and cost-effective evaluations of requirements.

The development of such a specialized quality model lies outside the scope of this thesis.

5.1.2 Oracle Forms version

A lot of the solutions mentioned in the previous chapter require Java integration. Oracle Forms supports Java integration from version 9i onwards. Legacy applications that have not yet been upgraded to this version must do so in order to be able to invoke Java. This upgrade may require substantial modifications to the application, as some methods and components have been deprecated and are no longer available[7].

7 See http://www.oracle.com/technology/products/forms/pdf/forms_upgrade_reference.pdf

Oracle 9i is also the first version to support Web Forms (see appendix A), which is a requirement for building more user-friendly applications (see section 4.4.4).

5.1.3 Isolation of reusable logic

As mentioned in section 4.4.2, logic which is part of an Forms application that needs to be invoked as (part of) a service must be isolated as a stored procedure in the database. Applications that have been designed as “fat clients” and do not feature strong separation of concerns often have their logic embedded in and distributed among the application's code (in the client). In these cases, the logic that needs to be reused first needs to be isolated and moved to the database tier to make it invokable (for example by wrapping them in a Java web service). This can require significant modifications to the legacy system's code, in which case the availability of good documentation of the system and the business processes it supports can be beneficial (see next section).

Applications in which reusable code has already been isolated into reusable libraries (within the application) only require minor modifications, as moving the code from the application to the database tier and encapsulating it as a stored procedure does not involve large modifications to the application.

Applications in which reusable code is has already been isolated and moved to the database as stored procedures do not require any modifications at all.

5.1.4 Documentation

In order to evaluate which approaches are feasible and how much effort it might take to implement them, documentation about the system is invaluable. Unfortunately, documentation is often outdated or not available at all, because architects, designers and developers are usually accountable and compensated for working software, and not writing and updating documentation.

Documentation can come in many forms, and depending on the approach, not all aspects might be important. On the lowest level, the application's source code itself should be documented to facilitate re-engineering and white box analysis techniques, which all require the code to be understood, often by developers that did not write the code themselves, or who have done so a long time ago and cannot remember the details of some particular piece of code.

On the next level, documentation about shared functions and libraries, the system's architecture (e.g. UML activity and class diagrams) and file layout, integration with other systems and the like are also essential for re-engineering projects. Since legacy applications are often monolithic in nature and lack cohesion and modularity (see section 2.4.2), changes in one part of the system can cause breakage in parts that were thought to be unrelated. Good documentation can help spot these relations, thereby facilitating re-engineering projects and making estimates of required effort.

The “highest” level of documentation consists of documents describing the sytem's (functional and non-functional) requirements, high-level architecture, etc. For example, these aspects might be documented and modeled with formal documents like UML use cases or ArchiMate models, or through informal written documentation. Documentation on this level is less important for re-engineering parts of the system, but all the more important for getting an overview of the system and the problems it solves, and the rationale behind its design and functionality.

5.2 SOA maturity

To measure an organization's “fit” to SOA principles and technology, a number of models and frameworks have been developed by academia, commercial IT vendors and other organizations. Among these are the the OASIS SOA Reference Model ([OASIS SOA]), the Open Group Service Integration Maturity Model ([OG OSIMM]), the CBDI SOA Maturity Model ([CBDI SOA]) and IBM's SOA Foundation ([IBM SOA]).

For the purposes of this thesis, Oracle's own SOA Maturity Model ([ORACLE SOA]) can be used to specify and measure requirements posed by leveraging approaches. The reason for choosing this model is that Oracle's consultants are already familiar with it, which makes the method easier to apply, and thereby more valuable to Oracle.

Oracle's SOA Maturity Model is similar to the Software Engineering Institute's (SEI's) Capability Maturity Model Integration (CMMI [CMMI]) framework. The CMMI is a process improvement framework, which defines a list of best practices and assigns an organization into one of five levels of “maturity” according to how many of the best practices are implemented in the organization. The model can be used to guide improvements across an organization by pointing out which missing processes should be implemented to reach the next level of maturity.

Oracle's SOA Maturity Model likewise defines five levels of “SOA maturity” (see figure 5.1 and appendix C), which describe how well the organization is suited to the SOA paradigm. Oracle's

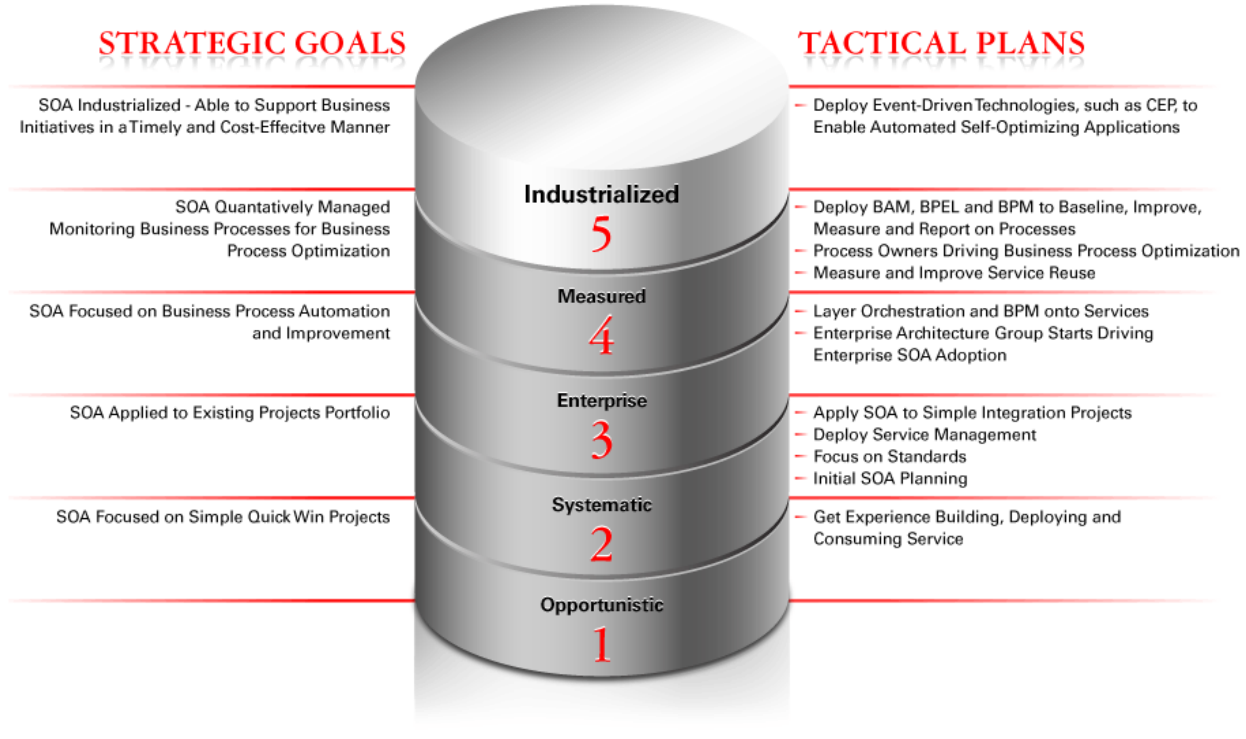
STRATEGIC GOALS
TACTICAL PLANS
SOA Industrialized - Able to Support Business Initiatives in a Timely and Cost-Effecitve Manner
Deploy Event-Driven Technologies, such as CEP, to Enable Automated Self-Optimizing Applications
Industrialized
5
SOA Quantatively Managed Monitoring Business Processes for Business Process Optimization
Deploy BAM, BPEL and BPM to Baseline, Improve, Measure and Report on Processes
Process Owners Driving Business Process Optimization
Measure and Improve Service Reuse
Measured
4
SOA Focused on Business Process Automation and Improvement
Layer Orchestration and BPM onto Services
Enterprise Architecture Group Starts Driving Enterprise SOA Adoption
Enterprise
3
SOA Applied to Existing Projects Portfolio
Apply SOA to Simple Integration Projects
Deploy Service Management
Focus on Standards
Initial SOA Planning
Systematic
2
SOA Focused on Simple Quick Win Projects
Get Experience Building, Deploying and Consuming Service
Opportunistic
1

Figure 5.1: Oracle SOA Maturity Model

model is mainly technology-driven, but includes organizational aspects as well. Like the CMMI, the SOA Maturity Model can guide an organization's progress towards service orientation and help define roadmaps for SOA adoption.

In the context of this thesis, Oracle's SOA Maturity Model can be used to specify requirements for leveraging approaches and measure an organization's maturity to evaluate the feasibility of that approach. For example, an approach that requires BPEL might require a SOA maturity level 2. An organization that wishes to implement this approach would first have to carry out a SOA maturity assessment to determine the requirement is fulfilled and the approach is feasible. Such an assessment is carried out by filling in a questionnaire with questions regarding various “Focus Areas” (see appendix D).

Chapter 6: Method

This chapter presents a method that helps organizations reach a decision on how best to proceed with their legacy systems. This method is the main contribution of this thesis.

The method is integrated into Oracle's methodology, the OUM. The next section will first introduce the OUM and describe the method's place within the OUM. Section 6.2 contains the description of the method.

6.1 Oracle Unified Method

The Oracle Unified Method (OUM) is an IT project framework based on the Unified Software Development Process (also called Unified Process or UP) and UML. Its goal is to help organizations "develop and implement technology-based business solutions with precise development and rapid deployment." [Oracle 2007] The solutions cover all of Oracle's IT products, from its database and middleware products to its suite of business applications.

The OUM adopts the iterative and incremental characteristics of the UP and employs its four-step approach (consisting of inception, elaboration, construction and transition) for project management. The method is designed with scalability and adaptability in mind in order to support both plan-driven (e.g. CMMI, Cleanroom and PSP) and agile (eXtreme Programming, Adaptive Software Development) software development methods.

The OUM includes three so-called Focus Areas (source: [Oracle 2007]):

- The *Manage* Focus Area "provides a framework in which all types of projects can be planned, estimated, controlled, and completed in a consistent manner."
- The *Envision* Focus Area "comprises the areas of the Oracle Unified Method framework that deal with development and maintenance of enterprise level IT strategy, architecture, and governance."

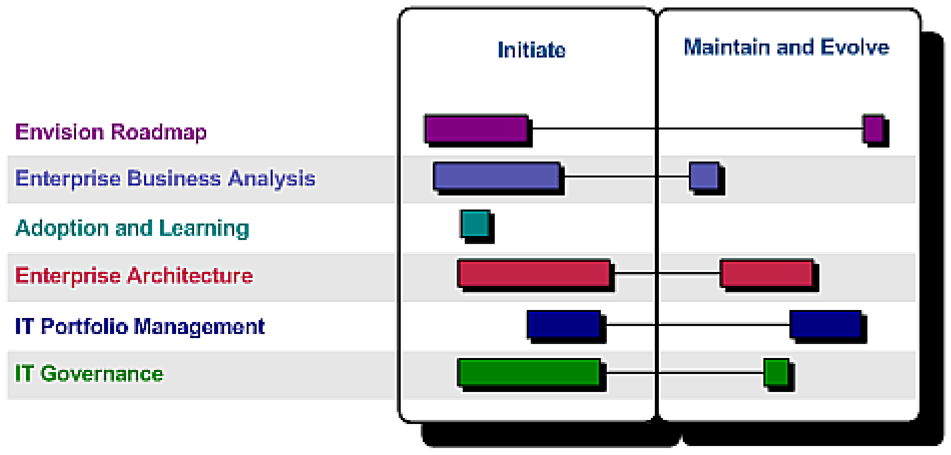

Figure 6.1: Overview of the Envision Focus Area processes and phases (source: [Oracle OUM])

- The *Implement* Focus Area "provides a framework to develop and implement technology-based business solutions with precise development and rapid deployment."

OUM divides projects into *phases* (each Focus Area has its own phases). In OUM, the most elemental unit of work is a *step*. Steps are combined into *tasks*, each of which has a specified output. Related tasks are grouped into *processes*. As an example, consider figure 6.1, which illustrates the phases ("Initiate" and "Maintain and Evolve") and processes for the Envision Focus Area.

In OUM, a *work product* is the output of a task. A 'work product' is explicitly distinguished from a 'deliverable' "to eliminate the risk of having the method deliverables confused with the contractual deliverables. A contractual deliverable is specifically referenced in the contract and often has a payment schedule attached to its acceptance." [Oracle OUM]

6.1.1 Place within OUM

One of the Envision Focus Area's objectives is to respond to critical business needs or pain points. As this Focus Area is concerned with an organization's enterprise-level IT strategy and takes into account its IT architecture and business objectives to develop and maintain strategic IT alignment, the method presented in this chapter fits naturally into this Focus Area.

The second process of the Envision Focus Area is Enterprise Business Analysis (see figure 6.1), which contains the task "Identify current architectural challenges" (EA.050). The goal of this task is to "review the current situation, and identify the current deficiencies related to that situation. In most situations, this view is on the enterprise level, but the scope engagement might also be limited to a part of the enterprise. Ensure that your effort is in line with the given scope. Also, the scope may be enterprise wide, but limited to a specific aspect of the architecture. For example, it may be limited to map the architectural aspects related to data structures, or only to cover security aspects of the architecture." [Oracle OUM] The method presented in this chapter is closely related to this task, as integrating legacy Oracle Forms applications in an SOA is usually considered an architectural challenge.

The next task is called “Identify architectural improvement options” (EA.060), which aims to “investigate further the current-state architecture and the ways in which it constrains the business [...] and determine possible options for improvement.” [Oracle OUM] The method presented in this chapter supports this task very well by proposing a way to develop a solution for a particular architectural challenge, namely integrating and leveraging a legacy Oracle Forms application in an SOA.

6.2 Method description

In the terminology of the OUM, the method presented in this chapter would be called an approach, and it would be carried out as part of a task (see previous section). The method description is structured according to the OUM task template. This ensures the method can be easily integrated into the OUM's Envision Focus Area material.

The method has been developed in collaboration with Jan Kettenis, an Oracle consultant who contributes to the OUM, particularly to the Envision material. His feedback on the method were an important part of its development, and ensured its fit into the OUM.

6.2.1 Overview

The premise of the method is that an organization has an Oracle Forms application that no longer supports one or more businesses drivers. This business driver is the starting point for the method. The method then directs experts to evaluate one or more possible solution alternatives to fulfill the requirement this driver poses on the legacy system.

The solutions provide a way to use or modify the legacy system in such a way that it (better) fulfills a desired requirement. This way, the existing system's value can be leveraged, and the business requirement can be fulfilled in the short term. Section 4.4 presents a number of these solutions. This list of solutions is not comprehensive, but it can (and should) be extended (see also the recommendations in section 8.3).

Each solution has some requirements that need to be met before it can be put into practice. Although there are a lot of different requirements, this method highlights two categories as especially important: the legacy system's quality and the organization's SOA maturity, both of which are described in chapter 5. If one or more of a solution's requirements is not met, several options are available. The first is to take the necessary steps to meet the requirement, for instance by improving a particular quality aspect of the legacy system. Another option is to evaluate other alternatives, if any are available. Finally, the last option is to consider replacement of the system (see section 4.5).

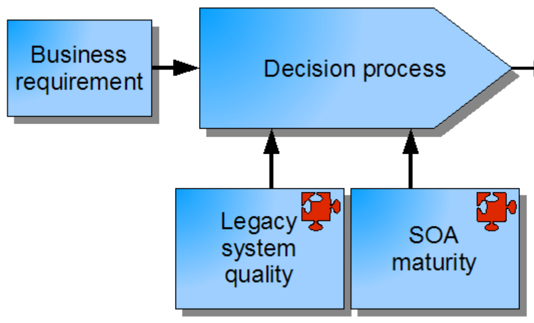

Figure 6.2: Method model

Figure 6.2 shows a graphical overview of the method's inputs and output.

6.2.2 Stakeholders and participants

In order to make informed choices at each step of the method, it is important to involve different stakeholders and experts on different domains. This section will discuss which knowledge has to be available to make decisions during the application of this method.

Since the method not only starts by identifying a business problem that needs to be solved, but requires decisions that can have long-term effect and which might require large commitments (budget, time, etc), it is essential to involve management at the strategic as well as business unit levels. To identify the requirement the problem poses on the legacy IT system and to find out why the system does not fulfill it, people with more detailed technical knowledge of the system and the organization's IT architecture should be involved in the discussion as well. All in all, a broad range of expertise needs to be available.

For the application of the SABA method (see section 4.6.3), Brooke and Ramage recommend that "[i]deally, a participant group should consist of about a dozen people and include:

- Senior directors (preferably including someone at Board level)
- Managers from different organisational functions (including Human Resources)
- IT specialists (preferably including a software engineer)
- Front-line staff (including those at the external customer interface)
- End users (preferably including an external customer)" [Brooke 2001]

The method presented in this section involves about the same expertise and is concerned with decision-making on the same organizational levels and comparable time scale, so the recommendation made by Brooke and Ramage is valid for this method as well. Since most steps in the method require a broad range of knowledge as well as authority to make decisions, it is beneficial to form a group of participants which are available most of the time.

Furthermore, the OUM also offers recommendations for selecting participants for its various processes. These should be taken into account for every process that is referred to by the method description.

6.2.3 Decision making process

This section describes how to carry out the steps that make up the method. An overview is presented in figure 6.3. The first and last steps are part of the method but are not described in detail here, since they lie outside the scope of this research, and are described in detail elsewhere in the OUM and other literature.

The right half of figure 6.3 shows the steps that need to be taken to apply the method in the form of an activity diagram. The left half shows which OUM processes and Focus Areas are related to each step.

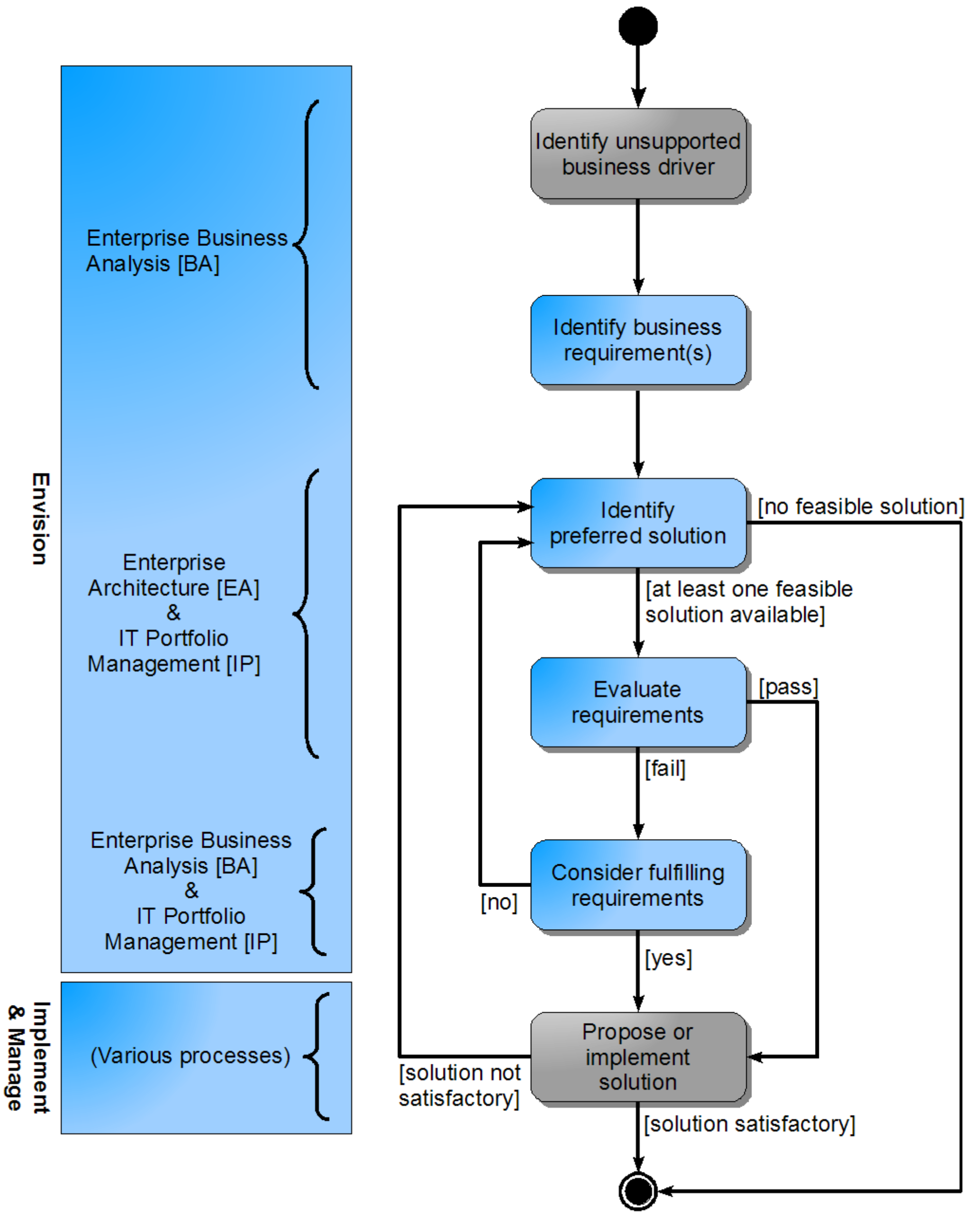
Envision
Enterprise Business Analysis [BA]
Enterprise Architecture [EA] & IT Portfolio Management [IP]
Enterprise Business Analysis [BA] & IT Portfolio Management [IP]
Implement & Manage
(Various processes)
Identify unsupported business driver
Identify business requirement(s)
Identify preferred solution
[no feasible solution]
[at least one feasible solution available]
Evaluate requirements
[pass]
[fail]
Consider fulfilling requirements
[no]
[yes]
Propose or implement solution
[solution not satisfactory]
[solution satisfactory]

Figure 6.3: Activity diagram for method steps
(right-hand side shows task steps; left-hand side shows related OUM processes)

A quick glance at the activity diagram shown shows that the method is iterative, just like the OUM itself. The SABA method developed by Brooke and Ramage (see section 4.6.3) is also “applied in an iterative way, so that technical options are tested out against the business needs. It is, thus, a dynamic tool which seeks to mimic the nature of organisational change, as far as is practicable.” [Brooke 2001] In order to reach a decision quickly, the method advises to first identify a preferred solution, and only then consider its requirements. If one or more requirements are not met, the organization can consider fulfilling them. If this is not desirable, a different solution should be identified, thereby iterating the process.

An alternative would be to first consider all possible requirements (for example by thoroughly measuring the legacy system's quality with all the metrics that have been presented in section 5.1) prior to evaluating the possible solutions. This alternative however would eat up more time and involve more people than strictly necessary, which would burden the decision making process, making it less agile, slower and costlier.

Identify unsupported business driver (prerequisite)

The starting point of the method is the realization that the legacy system under consideration does not fulfill a particular business driver. While identifying this business driver is a necessity for applying the method, this step is not really part of it. Rather, it should be considered a prerequisite.

In the OUM's Envision Focus Area, the Enterprise Business Analysis process contains a task called Identify Business Strategy (BA.010), which, as the name suggests, aims to identify an organization's business strategy. After identifying the strategy, the next step would be to evaluate which particular business driver is not supported by the legacy system.

Identify business requirement(s)

Approach

The next step is to analyze *why exactly* the legacy system does not support one or more particular businesses drivers. It is important to realize the difference between the business driver and the requirement it places on IT systems. The most common business drivers and their respective requirements have been presented in chapter 3 and are summarized in table 6.1. Although this discourse does not cover every possible business driver, it can serve as a foundation for evaluating requirements that have not been explicitly mentioned.

After identifying the business requirement that the legacy system does not fulfill, the participants should delve even deeper and discuss why the legacy system does not fulfill the requirement in detail. This analysis is useful when identifying the preferred solution.

For example, if the legacy system does not support the organization's business agility strategy, the first step would be to analyze why it cannot. One reason might be its lack of integration support, which prevents the organization from quickly changing processes because required modifications to the legacy systems interfaces with other systems take too long to change. A different reason might be its lack of flexibility because processes and are hard-coded into the application, which make them difficult to change. In these cases, the actual *business requirements* would be integration support and flexibility, respectively.

Business driver	Business requirement
Support for process orientation (BPR, BPM, BAM)	Business process support Flexibility Integration support
Business agility, service orientation	Flexibility Integration support
e-Business	Integration support Flexibility
Customer intimacy	User-friendly interfaces
Product leadership	Integration support Flexibility
Financial performance	User-friendly interfaces Maintainability Integration support Flexibility

Table 6.1: Summary of business drivers and corresponding requirements

The critical aspect of this step is to start with a business viewpoint to identify a problem with the legacy system. IT experts might be able to list number of problems and possible improvements, but what really matters is how the system's shortcomings hinder the organization from reaching its business goals.

Input

- Description of business driver(s) not (sufficiently) supported by legacy system

Activities

1. Identify unfulfilled business requirement(s)
2. Discuss in detail why the legacy system does not fulfill the requirement(s)

Tools and techniques

- Problem decomposition techniques
- Mind maps, brainstorms

Key participants

- Business experts and legacy application experts

Output

- Description of business requirement currently unsupported by legacy system
  - including the detailed technical reasons *why* it is not supported

Identify preferred solution

Approach

The next step is to choose a solution that will fulfill the business requirement. It is important to stress that such a solution need not increase the system's value for the long term. In fact, since it should be replaced in the medium to long term anyway, it will likely be difficult to get a return on large investments. The emphasis should be on finding a solution that adequately fulfills the business requirement, or which, in other words, “eases the pain” the organization currently has.

Chapter 4 elaborates on enhancing the value of legacy systems. Section 4.4 in particular presents a set of solutions which can enhance Oracle Forms applications in order to fulfill the requirements summarized in table 6.1. By no means should this set be considered complete, so new solutions can be considered too, of course.

The solution has to fit the organization's business and IT strategy (particularly any long-term plans for the legacy application itself) and its IT architecture. It also should also be realizable with the organization's IT developers' skills and. These aspects can be evaluated with various processes from the Envision Focus Area, most notably Enterprise Architecture [EA] and IT Portfolio Management [IP]. The Technical Scenarios Tool (TST, see figure 6.4) developed by Brooke and Ramage ([Brooke 2001]) can also help to make such a decision.

The OAR ([Bergey 2001], see section 4.6.1) and SMART ([Lewis 2006], see section 4.6.2) methods can help to evaluate the cost and technical feasibility of approaches that require parts of the legacy system to be componentized and wrapped as services (this approach has been discussed in section 4.4.2).

When some business requirements can be fulfilled in more than one way, a choice has to be made between the alternative solutions. To make this choice, the expected costs and advantages of each should be evaluated to find the most suitable approach. These aspects will often not fall into the same categories and hence cannot be compared easily. In such a case, it can be important to involve not only all the relevant stakeholders and experts, but also external consultants in order to make an informed trade-off.

In some cases, none of the possible solutions might deliver the necessary improvements, or they might all be infeasible to implement. For example, the required changes to the legacy system might

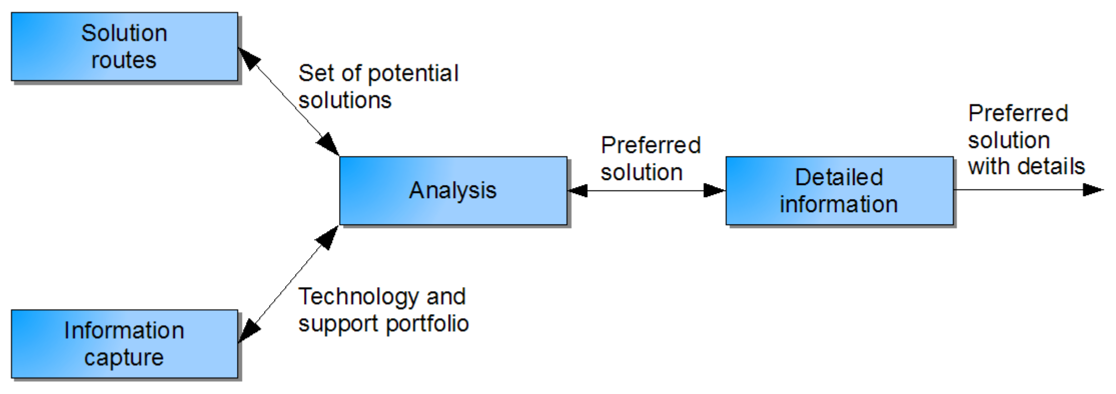
Solution routes
Set of potential solutions
Analysis
Preferred solution
Detailed information
Preferred solution with details
Technology and support portfolio
Information capture

Figure 6.4: Technical Scenario Tool overview ([Brooke 2001])

be so costly to implement that the costs outweigh the possible benefits. The solutions to these cases lie outside the scope of this thesis. One possible solutions is to replace the legacy system altogether, which is briefly touched upon in section 4.5.

Input

- Output of previous step (“Identify business requirement(s)”)

Activities

1. Consider (existing and new) approaches that can make the legacy system fulfill the business requirement
2. Evaluate effectiveness, feasibility and cost of solutions
3. Decide on preferred solution (if any)

Tools and techniques

- OAR (see section 4.6.1)
- SMART (see section 4.6.2)
- TST (see section 4.6.3)

Key participants

- Experts on legacy system
- IT architects

Output

- Proposal for legacy system modification approach

  OR
- No proposal because no solution is feasible or effective

Evaluate requirements

Approach

Once a solution has been selected, its feasibility has to be tested by evaluating any requirements the solution poses. In the context of this research, the organization's SOA maturity and the legacy systems quality are the two main areas that need to be considered.

The legacy system quality can be evaluated using different measurements. Some are informal (for example, the version of the Oracle Forms framework or the quality of its documentation), while others can be measured formally using software quality metrics. A general software quality model (based on the ISO-9126 standard) has been presented in section 5.1. Unfortunately, its metrics are abstract and very general, which makes the general model impractical for the purpose of this method. A quality model customized for the purpose of measuring attributes important for leveraging approaches would be very valuable. However, developing such a model lies outside the scope of this thesis (see recommendations in section 8.3).

The SOA maturity can be evaluated using Oracle's (or some other) SOA maturity model, which has been introduced in section 5.2. However, this knowledge is already available within Oracle's SOA maturity model, and in the literature in the case of the IT system quality metrics ([Fenton 1997], [SERC 2005]) and legacy reuse techniques ([Bennett 1999], [Sneed 2001], [Bergey 2001], [Lewis 2005]).

There are, of course, other requirements, like budget, time, and available knowledge and skills. While these will obviously have to be considered along with those mentioned above, this method does not consider these in greater detail because they are not unique to the particular problem this method deals with. The OUM's Envision Focus Area contains some processes that can help to make this evaluation (see the description of the previous task).

If all the requirements are already fulfilled, the solution can be implemented. If they are not, a choice has to be made as to whether or not to fulfill the requirements before the solution can be implemented or abandoned (see the next section).

Input

- Preferred solution proposal

Activities

1. Measure legacy system quality attributes
2. Measure organization's SOA maturity
3. Compare measurements to solution's requirements

Tools and techniques

- Oracle SOA maturity model
- Software quality model (including metrics)

Key participants

- Legacy system expert (for measurements of its quality)
- SOA expert on IT and organizational aspects (for SOA maturity assessment)
    - Can be external consultant

Output

- Conclusion on whether solution's requirements are fulfilled
    - Optionally: list of unfulfilled requirements

Consider fulfilling requirements

Approach

If the requirements for the preferred solution are not fulfilled, two choices are possible. The first is to take steps to fulfill the requirements, for example by improving the organization's SOA maturity or by improving a particular quality aspect of the legacy system. After the requirements have been fulfilled, the solution can be implemented.

Since it might take a long time to change the organization's SOA maturity or the legacy system to meet the requirements, it would prudent to re-evaluate the business requirement and the feasibility of the solution before implementing it. For example, business requirements might have changed, rendering the solution unnecessary, or a key developer familiar with the legacy system might have left the organization, which might render the solution infeasible to implement.

The second choice is to abandon the solution, if fulfilling the requirements is not feasible (the cost and implications of fulfilling the requirements do not outweigh the advantages that can be gained by implementing the solution), or does not fit the organization's strategy. For example, if the solution required the legacy system to be modernized where it was planned to be replaced out in the short to medium term anyway, making the investment might not be sensible. In case a chosen solution is abandoned, a different solution can be considered by iterating the method (see figure 6.3).

Input

- List of unfulfilled requirements

Activities

1. Evaluate fit (of fulfilling requirements) with business and IT strategy
2. Evaluate cost (in time, money, etc) of fulfilling requirements
3. Compare costs to expected benefits of preferred solution

Tools and techniques

(none)

Key participants

- Decision makers on strategic management level

Output

- Conclusion on whether requirements are to be fulfilled in order to implement solution

Propose or implement solution

Approach

When a suitable solution has been chosen and the organization meets all its requirements, the solution can be implemented. Since the chosen solution will most likely be new to the organization and its efficacy unproven, a small-scale prototype or proof-of-concept should be developed and evaluated to test the solution. This adds another iterative loop to the method, since the chosen solution may turn out to be infeasible to implement or unable to deliver the expected results.

The detailed description of the implementation of any of the solutions lies outside of the scope of this thesis. Several legacy modernization approaches and frameworks are described in the literature, for example the Options Analysis for Reengineering (OAR) [Bergey 2001] and the The Service-Oriented Migration and Reuse Technique (SMART) [Lewis 2005], which have been presented in section 6.1.

In the OUM, the implementation aspect is covered in detail by the Implement and Manage Focus Areas. However, before a solution is implemented, it should first be proposed as a “candidate project”. The Envision Focus Area contains a process called IT Portfolio Management (see figure 6.1), which “covers a ‘holistic’ view of the overall IT strategy of the enterprise. Its main purpose is to ensure that IT projects are aligned with the corporate strategy by maximizing the investment in IT projects while minimizing the risks.“ [Oracle OUM] One of the tasks in this process, called “identify candidate projects” (IP.020), verifies and prioritizes candidate projects. These tasks and processes should be carried out first in organizations that use the OUM's Envision Focus Area.

Input

- Preferred solution

Activities

- Propose implementation project

  OR

- Test solution using incremental development, prototype, proof-of-concept, etc

Tools and techniques

- Project management methods (SCRUM, PRINCE2, OUM, etc)
- Legacy reuse techniques (OAR, SMART)

Key participants

- Application developers (to test solution)

Output

- Implementation proposal

  OR

- Evaluation of solution feasibility (based on small-scale solution, prototype, proof-of-concept, etc)

End of method

At the end of the method, a solution need not necessarily have been found. In these cases, two options are available. If the “pain” caused by the fact that the legacy system does not support one or more business drivers is not too great, the organization could wait for a while and plan another iteration of the method in the near future, for example half a year or one year later. By that time, the context (business driver, budget, available developers) may have changed enough to change the conclusion. For example, new solutions might be available, or currently unfulfilled requirements might be met.

The second option is to consider replacement of the legacy system. Gartner recommends replacing Oracle Forms application in the medium to long term because maintenance and modernization will become increasingly difficult (see section 4.5). In cases where the legacy system already poses immediate problems which cannot be solved without large investments, its replacement should be considered as part of strategic IT Portfolio Management processes.

6.2.4 Project management

As is the OUM itself, this method is suited for both agile (e.g. SCRUM, see [Schwaber]) and iterative, plan-based (e.g. PRINCE2[8]) project management methodologies. Its iterative characteristics lean more towards agile methods however.

For example, the method can be applied according to SCRUM. In the pregame phase, the business requirement can be evaluated and a preferred solution can be identified. The solution's requirements are also evaluated in the “pregame” phase. A proof-of-concept or prototype of the solution can be implemented in a few sprints in the “game” phase. Depending on the result, the complete solution can be implemented according to SCRUM's “game” phase, or the next iteration within the method can identify a different solution.

The method can also be adapted to better suit plan-based methodologies. By evaluating the requirements before choosing a solution, it should be possible to identify a feasible solution in one or two sessions, without requiring iterations of the method just to identify a preferred solution. However, since the solution is likely to be different to other software development projects the organization has carried out, it would be a good idea to develop a proof-of-concept for the solution and select a different solution if it does not deliver the expected results, thereby iterating the method.

How best to apply and manage the method lies outside the scope of this thesis. However, the method's relatively simple structure and its integration into OUM mean organizations should be able to adapt the method to suit their preferred planning method.

6.2.5 Prerequisites

The first prerequisite has already been mentioned in the description of the first step of the method. The starting point of the method is an organization's business driver which the legacy system under scrutiny cannot fulfill. Identifying this business driver and translating it into an business requirement which the system needs to support is critical to the success of the method. The OUM contains two processes in the Envision Focus Area that can help to identify and formulate the business requirement. These processes are Enterprise Business Analysis [BA] and Enterprise Architecture [EA] (see figure 6.1).

The second prerequisite is an overview of the organization's current IT landscape. This overview should include its IT architecture and its IT portfolio. These aspects are important to review before a preferred solution can be identified, since the solution has to fit the organization's current and future IT architecture and IT portfolio. The OUM's Enterprise Architecture [EA] and IT Portfolio Management [IT] processes describe tasks that help identify and describe both aspects.

Additionally, the Organisational Scenario Tool developed by Brooke (see section 4.6.3 and [Brooke 2001]) can also be used. This techniques “begins with helping participants to describe their organisation as it currently exists. A useful starting point is an icebreaker exercise, such as asking participants to illustrate on paper their personal view of the legacy system” [Brooke 2001].

8 See http://www.ogc.gov.uk/methods_prince_2.asp

Another obvious prerequisite is documentation about the legacy system, which should include (among others): its (original) requirements (e.g. in the form of use cases), an overview of its technical design and operating environment, a description of its ties to other IT systems and an overview of its current use cases and end users. It is important for the system to be thoroughly documented in order to identify the “gap” between its “current capabilities” and its “target state” (in the terms of the SMART method presented in section 4.6.2).

6.2.6 Work product details

The output of the method is a decision on how to proceed with the legacy system. This decision preferably is a proposal for a technical solution to enhance a legacy system's value for the short to medium term. Such a solution can be proposed and evaluated as a candidate project and implemented. In some cases it might be necessary to first fulfill a requirement for the solution, for example by isolating business logic or enhancing the organization's SOA maturity.

Alternatively, if no feasible solution could be found, this too is an important result. This case warrants an evaluation of the replacement options for the legacy system, since it is now clear that the system cannot fulfill one or more business requirements and should be replaced in the medium to long term anyway.

An important byproduct of the method is the (documented) rationale behind the decision. It is important to reevaluate this rationale during the implementation of the solution, as changing business requirements may invalidate the solution (see next section). In case no solution could be found, the rationale provides a business oriented argument to replace the legacy system (as opposed to an argument derived from a purely technical viewpoint).

6.2.7 Critical success factors

The single most important success factor for the method is evaluating and testing the solution. As has been mentioned earlier, the solution will most likely be new for the organization, and its developers will have little to no experience with its implementation. To reduce risk, developing a prototype or proof-of-concept, together with incremental development of the complete solution is necessary to continuously evaluate the solution and its implementation, and to check whether the expected results are reached. Agile development methods are better suited to this approach than plan-based methods, although the latter too can be used by using iterative, incremental development.

Apart from reducing risk, incremental development also creates the opportunity to reevaluate the situation after each (or after a few) iterations. It is important not to loose sight of the initial motivation and rationale for implementing the solution, and to reevaluate its effectiveness when the context changes. For example, the business requirement might change, making the solution obsolete.

Another success factor is the participation of people that have enough knowledge (both organizational and technical) and the authorization to make decisions. Since the method is iterative, decisions should be made quickly in order to prevent long delays and to realize benefits early.

The discussion in section 4.5 posits that legacy systems should be replaced in the long-term. The method presented in this chapter is aimed at providing business benefits for the short-term by maximizing the investments in the legacy system. This means that small-scale and short-term solutions should be preferred to large-scale and long-term solutions, in order to minimize investments in the legacy system, and to minimize the organization's dependence on it.

Chapter 7: Validation

To validate the method presented in the previous chapter, it has been applied to one of Oracle's customers, Eurotransplant, to create a scenario. The first section of this chapter starts by introducing the organization and its business case and describes the application of the method to this case. The second section presents an evaluation of the method based on the validation criteria and the scenario.

7.1 Scenario

For the validation, an interview was conducted with Wilfred van der Deijl, IT architect at Eurotransplant in Leiden. Eurotransplant is a non-profit organization of about 90 employees. It is important to note that the organization has already chosen a solution to update its legacy application, and has already started implementing this solution. This means that the scenario does not start with a clean slate.

7.1.1 Introducing Eurotransplant

"The Eurotransplant International Foundation (Eurotransplant) is responsible for the mediation and allocation of organ donation procedures in Austria, Belgium, Croatia, Germany, Luxemburg, the Netherlands and Slovenia. In this international collaborative framework, the participants include all transplant hospitals, tissue-typing laboratories and hospitals where organ donations take place. The Eurotransplant region numbers well over 124 million inhabitants." [Eurotransplant]

Eurotransplant receives information about available organs directly from doctors from hospitals and manages waiting lists. As soon as one or more organs become available, Eurotransplant matches these against the patients on the waiting list. This process is complex and involves a lot of checks and exceptions. After a match is made, the organization facilitates communication and transportation between the donor's and receiver's institutions and arranges financial transactions. The organization is not involved with the procedures surrounding the transplantation itself however.

7.1.2 Business case

The legacy application

Eurotransplant has a moderately large Oracle Forms application, consisting of about 300 modules (or “screens”). This application was character-based in the 90s and was later migrated to a client-server setup using Citrix software.

In 2002, Eurotransplant started a modernization project to redesign large parts of the application because of their age. They evaluated whether to use Forms or JHeadstart[9], a newer development tool from Oracle for Java based applications. Since most users were more familiar with Forms interfaces, they chose to continue developing using Forms. Some users however were not familiar with Forms and preferred more modern web interfaces, so some parts of the application were rebuilt using web interfaces based on Java with JHeadstart.

In order to be able to use two different frameworks to create user interfaces, Eurotransplant decided to extract business logic from the older Forms applications and move it to the database, so both new application frameworks (Forms and JHeadstart) could make use of the same logic. About halfway through the redesign project, the developers and users started to see the benefits of the newer web interfaces (created with JHeadstart) and decided to shift from Forms to JHeadstart. Unfortunately, a big part of the application had just been rewritten using Forms, and the core parts of the system can't use a mixed approach, since having users switch between applications (with different kinds of interfaces) is just not practical, nor is it user-friendly. In 2004, the application was migrated to the latest Forms version in order to support Web Forms (web interfaces for Forms modules, see Appendix A).

In summary, Eurotransplant now had an application that used two types of interfaces and technologies. Although they started with the decision to use mainly Forms, about halfway it was decided a shift to web interfaces built with Java development tools would be preferable.

Business requirements

As has been discussed in section 3.4.1, the Forms interface is not considered very user-friendly, since it relies on keyboard shortcuts and does not provide very intuitive interfaces. To make matters worse, the Forms applications are (usually) designed for data entry, and do not guide or help the end user to fill in the information necessary for any particular task. The 300 modules are all accessible through one single menu, and the end user has to know which module to select. Since most of the application's end users are not regular users, the user-friendliness of the application is one of its

9 See http://www.oracle.com/technology/products/jheadstart/index.html

most important quality aspects. Hence **increasing the user-friendliness of the application** is an important objective, preferably taken on in the short term.

Eurotransplant has also already decided to migrate away from Forms altogether in the long term for two other reason. The first is that the current way of navigating the application is not only not user-friendly, it also does not support the allocation process, nor does it leave a good audit trail. Since the transparency and **accountability of the allocation process** are important for Eurotransplant, it was decided that the **application should support and guide the process**, which at the time was (and still is) done mostly on paper. After asking three consultancy companies for advice (one of which was Oracle itself), implementing BPEL to model and support the allocation process was chosen as the best solution. First, a proof-of-concept (POC) was developed to test the solution, after which one process (for one particular organ) was put into production.

Even with a successful prototype for the process-orientation requirement, how to get from the current situation (in which the application is used mostly for data entry) to the desired situation (in which the application manages the process and guides the user) was still an important but open question.

The second reason to move away from the Forms application is that its **monolithic nature makes it difficult to only offer parts of the application** to new users. Eurotransplant wants more countries to join its network. Most already have systems in place for all or most processes and do not want to switch to Eurotransplant's, or they want to do a small scale test before they join the network. Unfortunately, **the application is currently not modular enough** to provide only particular services to prospective members. Additionally, Eurotransplant would like to **offer services with optional user interfaces**. This would allow members' applications to either direct their users to Eurotransplants web interface, or use their own user interface and send the necessary information to Eurotransplant by calling a web service.

7.1.3 Method application

Identify unsupported business driver

The first step in applying the method is to identify the business driver the legacy does not support (anymore).

In the case of Eurotransplant, some (very important) users were having trouble using the application because its unconventional user interface. Furthermore, there were already plans to rewrite the application in the medium to long term because of the required modularity and process orientation mentioned above. The business drivers behind these “pains” are **customer intimacy** and **process orientation**.

Identify business requirement(s)

Input

Unsupported business drivers:

- Customer intimacy
- Process orientation

Activities results

The current legacy system does not support customer intimacy because standard Oracle Forms user interfaces are geared towards "power users" and take getting used to. A lot of the applications' end users do not use the application often and are not familiar with Oracle Forms user interfaces in general, and so cannot use the application efficiently. What is required is a new user interface, preferably web-based, which is more intuitive and user-friendly. Unfortunately, Oracle Forms applications can only use the native user interface technology.

As has been discussed in section 3.3.5, business processes embedded in legacy systems are often hard to change and not transparent. When business processes are not embedded in software, they usually are in people's heads, which makes these processes even less transparent, although they might be easier to change. These are exactly the problems Eurotransplant faces with their Forms application, and this is one of the reasons to implement BPEL.

Output

Unsupported business requirements:

- User-friendly interfaces (preferably web-based)
  - Currently not supported because Oracle Forms can only use "native" user interfaces
- Support for process orientation
  - Not supported because embedding processes in application is not feasible because of a lack of transparency and flexibility

Identify preferred solution

Input

See output of previous step: list of unsupported business requirements

Activities results

Eurotransplant decided that the ideal way to leverage the existing Forms modules while rewriting the application using the newer Java development tools (JHeadstart) would be to integrate the Forms interfaces in the new web interfaces. This would not only make interfaces more user-friendly because they could be rewritten using new web interface technology, it would also allow incremental migration from the Forms based parts of the application to the newer Java based framework, which was a medium to long term goal. This incremental migration meant that the most important user interfaces could be made more accessible in the short term, which would ease the "pain" soon.

To this end, Wilfred van der Deijl, in collaboration with Oracle developers, designed a mechanism[10] that enables communication between Java Server Faces (JSF) components and Forms modules. This

10 See http://www.oracle.com/technology/pub/articles/wilfred-adf-forms.html

in turn allows for (parts of) Forms modules' user interface elements to be almost seamlessly integrated in web interfaces developed on Oracle's Java application framework.

Evaluating the feasibility of the solution took a while because the underlying technology had to be developed and tested first. After initial tests and a prototype, the technique was deemed feasible to implement and able to deliver the required results, both from a technical and financial perspectives.

Output

The proposal to leverage the legacy application was to use the technology that allows parts of the legacy application's logic and user interfaces to be reused and integrated in a new application built using modern technology. In the long term, the old parts would be incrementally replaced by new software.

Evaluate requirements

Input

See output of previous step: proposal for leveraging approach

Activities results

The only requirement for this approach is that Forms had to be upgraded to a version that supports Web Forms (see appendix A), because older versions cannot integrate with the Java runtime and so cannot use the mechanism described above.

Apart from this requirement, isolation of businesses logic (see section 5.1.3) is beneficial for this approach because it allows this logic to be reused in the new application, which lowers the required development effort. One technique that can be used to achieve this isolation is discussed in section 4.4.2.

Eurotransplant's Forms application already fulfilled both of these requirements. The upgrade to Web Forms had already taken place earlier, and the isolation of logic had already been started because it would benefit the re-engineering project. This means that all requirements for the preferred approach were fulfilled.

Since the approach does not necessarily require an SOA, it does not place any requirements on the organization's SOA maturity.

Additionally to these requirements, the developers would need the skills necessary to develop using both sets of technologies (Forms and Java based). This trivial requirement was not explicitly considered as the developers mostly already had the necessary skills.

Output

All requirements are fulfilled.

Consider fulfilling requirements

This step was not necessary in Eurotransplant's case, as the requirements for the preferred solution were already fulfilled (see previous step).

Propose or implement solution

Input

Output of step "Identify preferred solution": proposal for leveraging approach

Activities results

The last step of the method is to either propose (and later evaluate) the solution, or implement it directly. This thesis recommends an incremental or agile approach to test the feasibility of the approach, and to reduce risk by first gaining some initial experience and evaluating the results before starting a full blown implementation project.

This is what Eurotransplant did, too. Since the key technology (which enables communication between Oracle Forms and Java) was brand new and not fully mature, the approach was "lab tested" first just to see if the technique could work in practice.

After the technology had matured and was ready for production, the first "quick win" project was started to test BPEL-driven processes and the integration of an "old" Forms interface in a new Java based part of the application. In this setup, the BPEL engine "hands out" so-called human tasks, which the end users can then carry out in a new (Java based) web interface that integrates and reuses an old Forms module's interface.

Output

This small scale project was finished successfully and demonstrated the feasibility of the approach and proved that the intended benefits could indeed be gained.

7.1.4 Work product (deliverable)

The method's work product (or end result) is the decision to continue the redevelopment of the main application based on Oracle's new, Java-based, application development framework, and to reuse the legacy application's logic and user interfaces by integrating these in the new application.

This approach has several benefits:

- It allows for *quick delivery of tangible business benefits*, in the form of user interfaces based on newer and more user-friendly technology for those users that do not use Forms regularly and therefore need intuitive interfaces.
- It also makes *incremental migration* possible.
  - This *reduces risk* compared to a Cold Turkey approach (see section 4.5),
  - but still allows for the *old technology to be completely replaced* in time.
  - Furthermore, the incremental migration *allows for prioritization*, which means those parts of the application that are most critical can be migrated first.

As a byproduct of the analysis of unsupported business requirements and the rationale for choosing this particular solution, Eurotransplant has gained some insight about requirements for the new application. Special care must be taken to ensure the new application's user-friendliness (to support customer intimacy), and modularity and flexibility (to support process orientation).

7.2 Evaluation

This section evaluates the validation criteria first presented in section 1.1.7, and which are reproduced in italics at the start of each of the following sections. Every criterion is discussed in a separate section below. The evaluation is based on interviews with Jan Kettenis, Oracle consultant and OUM contributor, and Wilfred van der Deijl, IT architect at Eurotransplant and speaker at Oracle's OpenWorld 2007 conference.

7.2.1 Innovativeness

“The method “must be innovative, solving a heretofore unsolved problem or solving a known problem in a more effective or efficient manner.” [Hevner 2004]“

According to Hevner et al. ([Hevner 2004]), design science should result in an artifact that is innovative. The method developed in this thesis fulfills this requirement in more than one way.

First of all, the literature contains few (if any) methods that apply to the specific problem this thesis addresses, namely leveraging a legacy system's value by finding a solution to make it **fulfill a particular business requirement in the short term**. Many existing methods (e.g. OAR and SMART, presented in section 5.6) take a technical viewpoint and assume a complete transformation of the legacy system is needed. Others (e.g. SABA) do take into account businesses issues but still consider large-scale and long-term solutions.

Apart from the literature, the method is also innovative with respect to the Oracle Unified Method, which at this point does not yet address legacy systems in its IT strategy processes.

7.2.2 Applicability

“The method needs to be applicable and complete. No steps or considerations relevant to applying the method should be ambiguous or missing.”

The second criterion for the validation is the applicability of the method. With respect to the scenario based on the Eurotransplant case, the **method did not leave out any relevant considerations or steps, and the level of detail in the description of the steps was found to be sufficient**. While the recommendations for the project group participants are not very detailed or strict, this is not considered a problem. Organizations usually already have enough experience creating project groups that the pointers given in sections 6.2.2 and 6.2.7 provide enough guidance.

It is important to note though that the Eurotransplant case has two shortcomings for this evaluation. The first is that the “preferred solution” originally took a very long time to mature. If the method would have been applied in the original context, the step “Propose or implement solution” would have taken more than a year, because the key technology necessary to implement the solution needed to be developed first. Since the method aims to help find a solution for the short to medium term, waiting this long to evaluate the feasibility of a solution means its problematic to even consider this solution. However, the method could have ended with the conclusion that no feasible solution was available *at the time*, but that it should be reiterated as soon as the key technology was mature.

The second shortcoming is that a solution has already been chosen, so the scenario is biased by hindsight. During the interview for the scenario, the step "identify preferred solution" only considered the particular technique that had already been chosen. Nevertheless, now that the solution is mature and publicized, organizations with a similar problem can (and probably will) consider this particular solution, and can reach the same decision as Eurotransplant did by applying the method.

7.2.3 Utility

The method must be useful to Oracle and its customers. This criterion is split into two sub-criteria.

Useful to Oracle

"The method has to fit both Oracle's existing methodologies and business strategy."

By relating the method's activities and goals to other parts of the OUM, and by describing it using the OUM task template, the method fits into the OUM's Envision material. Because of its focus on legacy systems, an area that as of yet is not thoroughly addressed by the OUM, the **method fills a gap in the OUM**.

Since there are so many Oracle Forms applications and many customers are looking to Oracle for a structured approach on how to leverage or modernize these, the addition of the method developed for this thesis to the OUM material increases the value of the OUM. Not only can this result in more contracts for consultancy services (which in turn can drive licensing and development revenue), it also shows Oracle's customers that it does not simply "abandon" Forms users, but is committed to help them leverage their investments and eventually migrate to newer technologies. This means the **method has commercial value for Oracle**.

Useful to customers

"The method has to help the subject organization reach a decision about its short to medium term strategy regarding the legacy application."

Business-oriented viewpoint

One "source" of the method's usefulness to organizations that have a legacy Oracle Forms application (note that by definition not all Oracle Forms applications are legacy) lies in its business benefit-oriented viewpoint. This viewpoint has two advantages.

The first is that the suggested **projects actually do support business requirements, and hence can show their potential for a return on investment** (ROI). This is in contrast with projects that result from a "technology push" viewpoint, where IT experts identify desirable IT projects from an

IT viewpoint, where the intended business benefit is often unclear or difficult to “sell” to decision makers.

The second advantage is that **projects can be started quickly because management is already involved** (as participant during the application of the method) in the decision-making and knows which problems are solved. In turn, this means **business benefits can be delivered quickly**.

There is even a third, though indirect, advantage. The business-oriented reasoning about a legacy application's shortcomings with respect to business requirements actually **helps identify requirements for new (replacement) applications**. In a situation where no solution could be found to make the legacy system support a particular business driver, the organization at least has gained some knowledge about which requirements it should pose on any new application it acquires or develops.

Facilitate SOA adoption and migration or replacement of legacy systems

By emphasizing solutions that are as less invasive as possible to support particular business drivers in the short term, the method helps organizations disinvest in legacy applications and and find ways to replace these applications in the long term. This thesis also advocates incremental approaches because they can reduce risk compared to Cold Turkey approaches. In this way, the method can facilitate SOA adoption, which is believed to provide numerous advantages (see sections 3.3 and 3.4) and provide a better strategic fit between IT and business strategy (see section 3.2).

Chapter 8: Conclusions and Recommendations

The goal of this research was to develop a method that helps organizations that face the problem of legacy applications not fulfilling business requirements make a decision about how to deal with this problem. The main research question was:

How can existing legacy IT applications be leveraged in a Service-Oriented Architecture, and which factors influence the choice between possible alternatives?

The result of this design research is the method that is described in chapter 6. This novel approach takes a business-oriented viewpoint and helps organizations identify a solution that leverages the legacy application to fulfill a particular business requirement.

The method fits nicely into the Oracle Unified Method (OUM), which increases its value for Oracle. Its applicability and usefulness have been validated by a creating scenario based on a real business case.

This chapter presents the conclusion of this thesis. The first section provides answers to the research questions posed in the introduction. The next section discusses questions that have not been sufficiently answered and proposes future research. Finally, the last section presents recommendations to Oracle and its customers.

8.1 Answers to research questions

This section provides brief answers to the research questions posed in the introduction.

Which business drivers are better supported by SOA than by legacy applications, and why?

To answer this question, a list of common business drivers was compiled based on two frameworks found in the literature. The next step was to analyze which requirements these business drivers pose on IT applications, and which of these requirements are better supported by SOA and how. Table 8.1 summarizes the business drivers and their respective requirements.

Business driver	Business requirement
Support for process orientation (BPR, BPM, BAM)	Business process support Flexibility Integration support
Business agility, service orientation	Flexibility Integration support
e-Business	Integration support Flexibility
Customer intimacy	User-friendly interfaces
Product leadership	Integration support Flexibility
Financial performance	User-friendly interfaces Maintainability Integration support Flexibility

Table 8.1: Summary of business drivers and corresponding requirements

How can legacy IT applications be leveraged in an SOA to better support business drivers?

Generally speaking, the answer is to modify the legacy system just enough to make it support the requirement (see table 8.1) the business driver poses. Several approaches have been identified, each of which deals with one (or more) of the requirements. Among these are:

- Isolating and wrapping business logic to a) componentize applications to *increase maintainability* and b) *support integration* by providing services through standards-based interfaces
- Interface with and call BPEL engines to *support business process orientation*
- Integrating legacy applications into new user interfaces to *increase user-friendliness*

Note that the list of approaches is most likely incomplete, which means the research question is only partially answered. Creating a comprehensive list of leveraging approaches is an unrealistic goal however, since some problems will most likely have unique solutions. The focus should be on gathering reusable approaches and techniques. One of the recommendations made in the next section is to create a "library" of approaches.

Which requirements do leveraging approaches pose on an organization's SOA maturity and the legacy system's quality?

This research has not provided a general answer to this question. A more thorough answer to this question can only be presented after specialists have created a “library” of approaches (see next section) which includes detailed descriptions of their requirements.

We can conclude that every approach can pose unique requirements, although some requirements are probably more common than others. For example, isolation of business logic and maintainability should be important for almost every conceivable approach that requires modification of the legacy system.

How can an organization's SOA maturity be measured?

Numerous SOA maturity models and frameworks are described in the academic and industry literature (see pointers in section 5.2). Many of these include “assessments” or measurement approaches. For Oracle customers, Oracle's SOA Maturity Model is an obvious choice.

Different models (or frameworks) have different viewpoints however, and so focus on different aspects of an organization's SOA maturity. For example, Oracle's model is technology-oriented, so using a more organizational-oriented model might result in important alternative (or additional) outcomes. In the context of this research however, a technology-oriented SOA maturity model is a good choice, since it is used to evaluate the feasibility of leveraging solutions, which pose requirements that are more technical in nature as well.

Which quality attributes of legacy systems are relevant with respect to this thesis?

Section 5.1 presents a list of attributes that can be important for different leveraging approaches. In short, maintainability, isolation of business logic, the version of the Oracle Forms framework, and detailed documentation have been found to be relevant for the leveraging approaches considered in this thesis.

It is important to note that this list is based on the incomplete list of approaches presented in chapter 4, and thus should itself not be considered complete. More quality attributes might be identified as relevant after more approaches have been considered.

Subquestion: How can these quality attributes be measured?

This research has investigated a software quality model which could be used to measure quality attributes, but found it too general to use in the context of the decision support method presented in this thesis.

One of the recommendations made in the next section is to develop a specialized software quality model, together with software tools to assist in making measurements, to help define and measure relevant quality attributes.

8.2 Open questions / future research

8.2.1 Business drivers and leveraging approaches

The answers to the research questions presented above have already pointed out that this research has not presented a definitive answer to the questions which business drivers are better supported by SOA than by legacy applications and how legacy applications can be leveraged to support these business drivers.

Chapter 3 investigates the first question by first using two frameworks to identify the most common business drivers (see table 8.1). The next step is to identify which of these drivers are promised to be better supported by SOA, and why. There are two issues with this approach, both of which pose opportunities for further research. The first issue is that there are more business drivers than have been identified. These could be uncovered by a thorough literature study. The second issue is the discussion of how SOA can better support these business drivers. First of all, since not all business drivers have been identified, this discussion is not complete. More importantly though, this research does not provide any empirical evidence that SOA does indeed support business drivers better than (most) legacy applications. Now that SOA adoption is becoming increasingly common, it should be possible to gather this data and show which promises SOA can fulfill (and under which circumstances), and which it cannot.

The second question has been dealt with in chapter 4. The approaches that have been gathered by conducting interviews with experts and asking them how the business drivers could be supported by modifying a legacy system. The first problem is that the list of business drivers is not complete, as is pointed out above. The second problem is that even for the business drivers that were identified, not every possible leveraging approach has been found. One of the recommendations made in the last section of this chapter is to collaborate on building a "library" of approaches (see section 8.4.2).

8.2.2 Level of detail and comprehensiveness of requirements

The discussion of requirements relevant for leveraging legacy systems is based on the leveraging approaches identified by conducting interviews with Oracle experts. Since this list of approaches is certainly not comprehensive, and since the approaches' requirements have not been studied in detail, it should be further investigated whether all the requirements relevant for *any* (or at least most) approach(es) have been considered.

The only two areas that have been considered in chapter 5 are the legacy system's quality and the organization's SOA maturity. Other requirements, like budget, time, skill of developers, have only been briefly mentioned in the method description.

Further research is necessary to either identify other relevant requirements, or to show that the requirements considered by this thesis are indeed sufficient.

8.2.3 Method validation

The method has been validated using only one scenario. To really establish its applicability and its usefulness, some real case studies should be conducted with the a few of the many organizations (thousands, according to [Gartner 2007]) that have a legacy Oracle Forms application and are adopting SOA.

8.2.4 Generalize or specialize method to support similar problems

The method can be generalized to work with any legacy application in any IT architecture. Some aspects, for example the requirements that need to be considered, will differ for other contexts, but the short-term and business-oriented viewpoint is applicable to similar problems.

8.3 Recommendations

This section concludes the thesis with a few recommendations for Oracle and its customers.

8.3.1 Create specialized quality model

To enhance the value of the method and increase its utility, Oracle should develop a small and specialized software quality model for legacy systems based on Oracle Forms technology. Such a model, together with software tools that automate and / or facilitate inspection of application code, could help any decision making process that requires significant modifications to an Oracle Forms application by providing relevant information about its structure, quality and architecture. Section 5.1.1 has already touched upon such a model based on the ISO 9126 standard.

For example, one measurement for cohesion could be to count the number of references to a particular database table in each module. In a system with high cohesion, the number of modules would be small (ideally just one module), which makes it is easier to maintain and modify.

8.3.2 Share knowledge about leveraging approaches

The leveraging approaches presented in section 4.4 have been gathered by interviewing Oracle specialists. Every specialist described at least one approach that had not yet been mentioned by other specialists. Apparently, specialists do not yet systematically share knowledge on how best to leverage Oracle Forms applications in particular situations.

The second recommendation thus is to create some sort of “repository” or “library” to share knowledge about leveraging approaches between Oracle developers, customers, and partners. Such a repository would support the decision making process and implementations of particular solutions by providing a list of possible leveraging approaches, case studies, solutions to technical problems, best practices, and other relevant information.

8.3.3 Replace Oracle Forms applications sooner rather than later

As has already been mentioned in section 4.5, Gartner recommends organizations to “[a]pproach Oracle Forms within a 'containment' strategy [and] migrate to industry-dominant technologies [...] to align with industry best practices in the future.” [Gartner 2007]

The method presented in this thesis aims to mitigate business “pains” caused by legacy systems in the short to medium term. The definition of legacy used in this thesis (see section 2.1.1) stresses that outdated technology alone does not make a system “legacy” – what does is its inability to change to support requirements. In the long run however, virtually every Oracle Forms application will fit this definition of legacy, because it will be increasingly harder to find developers experienced with the technology, which in turn will make maintenance and modification projects more time-consuming and increasingly expensive.

Hence, the final recommendation of this thesis is in line with Gartner's. Applications built on Oracle Forms should be replaced sooner rather than later, because postponing the replacement will increase the cost of it, for the reasons mentioned above.

Unfortunately, it is difficult to estimate the cost of immediate replacement, let alone estimate the cost of replacement in the future. This makes it hard to make a decision on financial data alone, since the return on investment (ROI) is almost impossible to predict accurately. Furthermore, since most applications still fulfill most requirements, “preemptive replacement” is hard to justify.

Appendix A: Introduction to Oracle Forms

Architecture

Oracle Forms is based on a two-tiered architecture (see figure A.1, left-hand side), with a database to make up the data tier and “fat client” applications for the application tier.

Recent versions of Oracle Forms also support an alternative architecture (called Web Forms), which introduces a third tier (see figure A.1, right-hand side).

Data tier

The data tier is implemented on Oracle relational databases which contain all the enterprise's data. These databases offer advanced functionality that goes beyond reliable data storage, some of which are used by Forms applications.

Constraints

One of the more common database features are constraints, which allow administrators to specify constraints on the data stored in the database to enforce relational integrity and business rules. For example, it is possible to enforce that every order must refer to an existing customer, and its “data payable” must be later than the date the order has been entered.

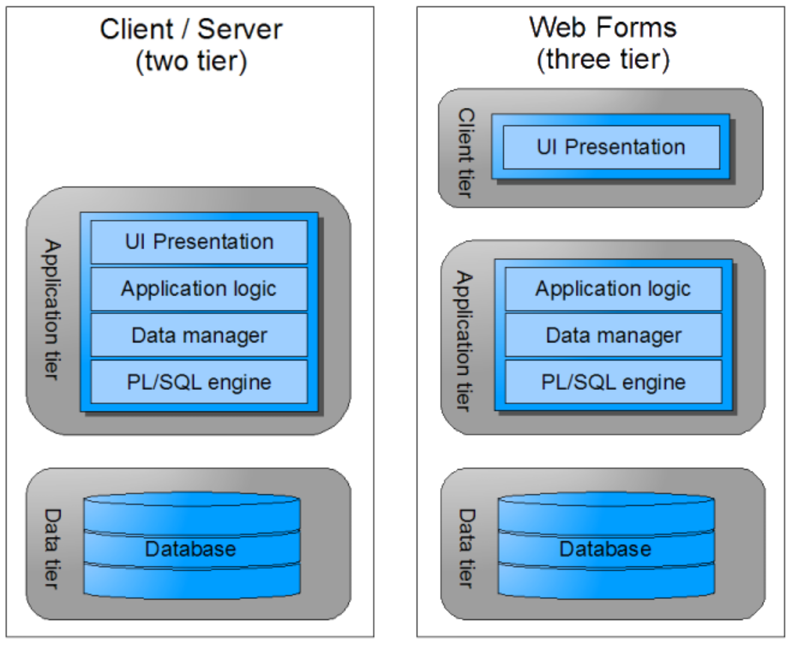

Figure A.1: Oracle Forms architectures

Stored procedures and triggers

Stored procedures are blocks of PL/SQL[11] (Oracle's proprietary procedural extension to SQL) code that can be called by applications. This enables reuse of code and modularization, much like a function in most programming languages. When a stored procedure is associated with a database table or an event (such as the insertion of a new tuple), it is called a trigger.

Application tier

The application tier is implemented in (one or more) Oracle Forms client application(s). These applications are divided into modules, each of which is (usually) tied to a particular business object, like customers, orders, employees, etc. Each module presents an interface for dealing with a particular object, which ties the module tightly to the database table(s) the object is stored in.

Each module contains an interface and all the logic required to control the interface and perform data processing and other logic. All this logic is written in PL/SQL. The application itself is divided into modules, each of which contains a proprietary interface description which is interpreted by the Forms runtime engine to render a user interface, interface logic (implemented in PL/SQL) to control the interface, and application (and business) logic that processes data and sends ad-hoc SQL queries to the database or calls stored procedures.

To navigate the application, the user can use a menu that displays all modules accessible to her, or let a module's interface logic open a new module.

While Oracle recommends keeping the clients as “thin” as possible by implementing as much application logic in stored procedures in the database, customers do have a choice and some have made applications that store the majority of the application logic in the client modules.

Client tier (Web Forms only)

Forms applications can be web-enabled by migrating them to a recent version of Oracle Application Server (version 9i onwards), which will interpret the module code and send user interface code to the client browser, which in turn renders it using a Java Applet. While this approach does make it possible to make the application accessible over the web, and without installing the client software, it does not create a separate presentation tier, since the interface logic and the application logic are still integrated in the same pieces of code and are tightly coupled.

Integration

In order for an external program to interface with an Oracle Forms application, it has to connect to the database at runtime and query it directly using SQL statements (or invoking stored procedures).

11 See http://www.oracle.com/solutions/application_development/pl_sql_dev.html

There are several problems with this approach. First of all, the database has to be “open” to connections from outside of its network, possibly even outside of the company's network. This introduces maintenance and security issues.

Another problem problem with this approach is that the developers of the external program have to use proprietary software to connect to the database and learn a proprietary language (or “dialect” of SQL) to issue correct SQL statements (unless they are already familiar with Oracle's SQL dialect).

Direct database connections are also highly undesirable from a maintenance perspective, because the external program (which is usually developed by different people) has to be changed when the database changes.

Appendix B: Introduction to SOA

Architecture

Service Oriented Architectures follow a multi-tier (also called n-tier) approach to system architecture. Instead of dividing the application in two layers, there are numerous layers that each deal with a particular aspect. For a detailed discussion see [Alonso 2004]. [Papazoglou 2005] and [Kontogiannis 2007] present more abstract overviews of the concept and research issues.

An SOA intrinsically follows a multi-tier architectural approach, which divides the application across different layers. When used correctly, this approach results in lose coupling, which simplifies maintenance and improves scalability, because each layer can be modified or extended without affecting the others.

Dividing software into manageable and independent pieces serves the purpose of increasing maintainability, since changes in a service do not affect other services (as long as the Application Programming Interface does not change). It also allows changes to be made much more rapidly to quickly align IT support with changing business needs.

Data tier

The “lowest” tier, the data tier, is very similar (or even identical) to its counterpart in legacy architectures. It is made up of a database that stores data and offers APIs (like JDBC, ODBC and proprietary libraries) that can be used to issue SQL statements or invoke stored procedures to query and modify the data. The only difference is that while the database *is* capable of storing and executing stored procedures and managing data constraints, SOA purists would keep these out of the data tier and move them to the upper tiers in an SOA. Nevertheless, (relational) legacy databases can usually serve as data tier for SOAs with minimal or no adjustments, since the same software and data models are used in both cases.

Middle tier

Even though a different number of logically independent layers between the data and presentation tier can usually be distinguished in an SOA, these are commonly referred to as

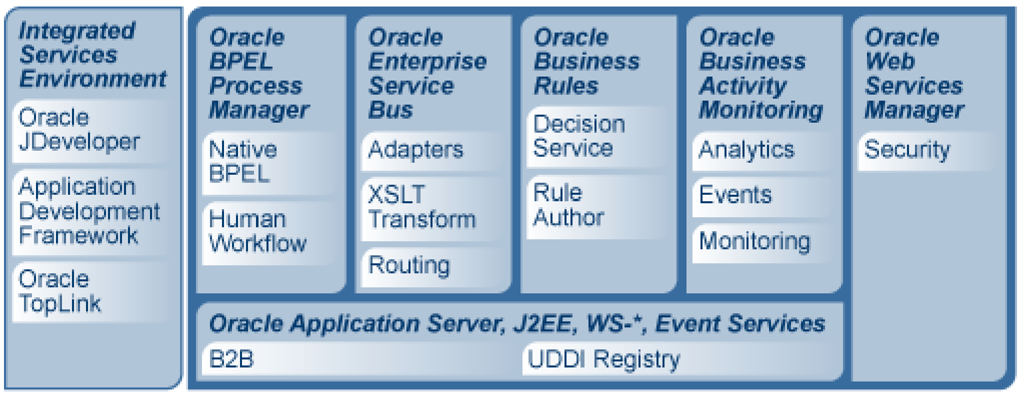
Integrated Services Environment
Oracle JDeveloper
Application Development Framework
Oracle TopLink
Oracle BPEL Process Manager
Native BPEL
Human Workflow
Oracle Enterprise Service Bus
Adapters
XSLT Transform
Routing
Oracle Business Rules
Decision Service
Rule Author
Oracle Business Activity Monitoring
Analytics
Events
Monitoring
Oracle Web Services Manager
Security
Oracle Application Server, J2EE, WS-*, Event Services
B2B
UDDI Registry

Figure B.1: Oracle SOA Suite's middle tier (source: Oracle Corporation)

the middle tier. This layer is heavily supported by middleware, which consists of different pieces of software that provide services like message delivery and queuing, transactions, service registries and more (see [Britton 2004]). Figure B.1 depicts Oracle SOA Suite's middle tier.

The actual application logic (and some business logic) is implemented in services, which are "course-grained, discoverable software entit[ies] that exists as a single instance and interact with applications and other services through a loosely coupled (often asynchronous), message-based communication model" [Brown 2002]. Services can be written in any programming language, as long as they can be invoked using standards (see section "Integration" below). This means that a service consumer does not have to know (and hence does not need to make any assumptions about) how a service is actually implemented, which reduces coupling.

To support actual business processes, several services have to be invoked according to complex flows (which can include constructs like branches, parallel execution and loops, among others), where the output of one service may be used as input for the next. This process of "service orchestration" can be modeled using the Businesses Process Execution Language (BPEL). BPEL engines can then execute these models and automatically invoke services and wait for asynchronous responses from other services. Each BPEL process can itself be offered as a service and used in other BPEL processes.

The middle tier contains application servers, which can be used as deployment platforms for both services as well as the BPEL engines, as well as numerous other middleware software pieces like ESBs, message queues, transaction processors, etc (see [Britton 2004]).

Presentation tier

The "uppermost" tier is the presentation tier, which manages channel-specific device dependencies (like communication protocols and user interfaces). For example, the presentation tier is responsible for creating an HTML web page which it sends over HTTP to an end user's browser, or it may create a voice message which is sent using VOIP or email. The message's particulars (like contents, recipient) are specified by the services in the middle tier – the presentation tier only only transforms the message into a device-dependent form and sends it using the right protocols.

The presentation tier is also responsible for receiving input from different user channels, for example an HTML form submission. These events are forwarded to the middle tier, which can react on these, for example by starting new BPEL process instances (in the event of a customer ordering a product using the company's website).

Integration

Integration is one of the cornerstones of SOA. While direct access to the database (as described in section 2.4.3) is still technically possible, SOA promotes the use of novel interfacing technology, based on open standards and the web. These standards include WSDL, UDDI and SOAP, and communication takes place over HTTP. [Alonso 2004] provides a detailed for a description of these standards. The advantages of these standards are that the problems described in section 2.4.3 are mitigated or eliminated. See section 3.3.2 for more details.

Appendix C: Oracle SOA Maturity Model

Level 1: Opportunistic SOA

Level 1 organizations are early in their adoption of SOA. They recognize the potential and have begun to work *opportunistically* to gain *early success* to garner additional support.

SOA knowledge is often contained within a small group in the IT organization, possibly spread amongst several project teams. These groups are *exploring* technology and gaining valuable *education* and *experience* with SOA technology.

IT teams may be using Web Service technology to *share information* between systems. The teams have recognized the SOA value of technology standards and the opportunity for widespread sharing and reuse of their efforts.

Level 1 is about building a foundation and their efforts are focused on getting systems to talk, using standards to enable interoperability. They are working tactically to solve immediate project challenges with little to no thought about larger enterprise SOA adoption.

While having limited enterprise SOA thinking, level 1 organizations may endeavor to complete projects that include implementation of new SOA technology. These projects may be of considerable enterprise value and may produce significant direct returns to the business. However, without the proper support in place across other SOA dimensions, the projects likely will produce fewer long-term enterprise SOA assets than desired.

In summary, level 1 organizations focus is on technology *standards*. The scope of efforts is narrow, likely a single project or application. Some thinking may reach to an organization. Ownership of SOA assets remains with the immediate project team. They are the producer and consumer of most assets. This often will result in less clear service specifications since services are produced and consumed within the team. Abstraction is often low as the team is building foundation services that wrap existing applications. It is difficult with these first projects to reach high-levels of abstraction and produce well-defined reusable business services.

Level 2: Tactical SOA

SOA Level 2 is about *expanding the foundation service portfolio* and creating new services through *service collaboration*. Level 2 organizations are taking steps to expand towards enterprise SOA, while tactically using SOA on everyday projects. Services respond to events and architects are planning the event portfolio alongside the service portfolio.

At level 2, organizations are beginning to benefit from reuse of existing core foundation services. As their service portfolio is expanded, they are not only sharing data but now *sharing functions*. As data services expand, the organization benefits from a single point of access for data elements. As functions are shared, the ability to share business rules across multichannel applications grows. Each of these improve quality, consistency for service consumers.

In contrast to level 1 where the SOA scope is narrow and application integration is often point-to-point using new SOA technology, level 2 projects typically transition to more complex integration solutions. These may include (but are not limited to) employing an enterprise service bus, performing transformation across application and canonical models, and including more than two applications in the integration scenarios.

Knowledge and experience are spreading as SOA is used across more projects. The operations of SOA deployment environments expand from the core early adopter team to the broader operations team.

In summary, level 2 organizations focus is expanding and collaborating. This involves expanding the service portfolio, identifying key events for which services must respond, and building new services through service collaboration.

Planning, knowledge, and experience are also expanding. IT is reaching beyond its walls to collaborate with business analysts. Business analysts are preparing process models for the important transition to level 3. As the service portolio expands, the information model is being clarified.

Level 3: Strategic SOA

SOA Level 3 organizations are creating visible *business* impact. They have a well-defined and deployed service portfolio available for reuse. Service orchestration is being used to *automate business processes* which improves end-to-end processing time, increases productivity, and reduces errors previously occuring within manual activities. Automated processes are responding to, and creating new business events that enable monitoring to create new opportunities for business visibility.

The scope of SOA activities has expanded to multiple organizations within the enterprise. The portfolio includes services at the business service layer of the layered reference model enabling not only sharing data and functions but sharing processes. IT is working with business analysts to ensure that these business services are a clear digitial representation of actual business process activities.

The level 3 organization has a reference model for *deployment of shared services* enabling evolution of hardware silos to more flexible service-based, shared hardware usage.

The organization is also advancing along non-technology dimensions. Processes and governance mechanisms are established for *funding* development of shared services, the *ownership* of which is now being defined. In addition, informal internal SOA training activities are ongoing as well as formal *training* to spread SOA knowledge across the organization.

Level 4: Enterprise SOA

SOA Level 4 organizations continue to master their skills around business process automation but go a step further to *quantitatively manage* their business services. As part of this evolution, the focus now shifts to *monitoring, measurement and improvement* across all levels of the service-enabled business.

Level 5: Industrialized SOA

SOA Level 5 organizations are seen as industry leaders, know for their continuous innovation and world-class business processes. They enjoy a well-developed service ecosystem toward which the majority of their customers, partners, and suppliers have ultimately gravitated. They employ broad and flexible business-to-business (B2B) service models as well as numerous interaction channels for their expanding business-to-consumer (B2C) relationships.

Appendix D: Oracle SOA Maturity Assessment Questionnaire

Focus Area		1	2	3	4	5
ARCHITECTURE	**Question**					
Role of Enterprise Architecture	How would you describe the role of architecture in your organization?	We have some IT people who call themselves architects, but no formal architecture team.	We have a dedicated architecture team, but their impact on the enterprise is not yet well understood.	We have been investing heavily in enterprise architecture, but we are only beginning to realize its benefits.	Our enterprise architecture team continuously proves its value through IT metrics and business results.	Our enterprise architecture team has transformed IT into a growth engine for the business.
Cataloging and Reuse	To what extent are services cataloged and reused in your environment?	We are still just experimenting with web services, and thus haven't begun cataloging or reusing them.	There is no formal catalog of services, but reuse is still happening through informal communication channels.	We have a formal and accepted method for registering and finding services in our environment.	We enforce the use of registries or repositories at both design time and runtime, and we carefully monitor the reuse of services.	We are advanced users of registries and repositories, using both centralized and federated models.
Planning and Guidance	How extensive is the planning and guidance provided by your architectural team?	Our architects are an informal group and aren't really involved in planning.	Our architects offer good advice, but it is hard to implement their ideas, so we rarely do their bidding.	Our architects have a fair degree of authority, and they have visibly contributed to the success of a few key projects.	We have a centralized team of architects who are involved across every line of business.	Our chief architect has a seat at the board room table.
INFRASTRUCTURE	**Question**					
Standards	How would you characterize your organization's commitment to standards?	We have not yet adopted a formal stance regarding any particular standards.	We have begun to rationalize our IT assets into what we consider standards-based versus non-standard.	We have a clear understanding and a formal position on which standards are important to our business and why.	The adherence to standards plays a critical role in the planning and budgeting process for any and all IT projects.	We actively invest in emerging standards that we feel may impact our business down the road.
Security and Monitoring	How do you secure and monitor your data and business logic today?	Most of our systems have their own separate security and monitoring models, and little has been done to unify them.	Security is addressed on a case-by-case basis as services get shared for the first time.	We have just begun to centralize identity management into its own layer of abstraction, and we are standardizing the security models for different service types.	Centralized identity management is fully deployed, allowing for greater flexibility in access control provisioning, compliance, and reporting.	We have completely abstracted security, identity, and monitoring operations out of our existing systems in order to simplify development and centralize control for compliance purposes.
Management and Operations	How would you characterize the management and operations environment in your company?	Most of our systems have their own separate management and operations consoles, and there is little consolidation among them.	We have developed our own unique set of automation techniques to make our systems more manageable.	We have invested in management tools to bring IT systems under a common management platform, but coverage is not complete.	Our management platform is fully deployed and offers reasonable coverage, but we are still incapable of quickly providing detailed root cause analysis.	Management and operations are consolidated into a unified console, allowing for easy troubleshooting and monitoring of SLAs and compliance requirements.

Focus Area		1	2	3	4	5
DELIVERY	**Question**					
Project vs. Enterprise Focus	How does your IT team typically fulfill requirements for a new application?	Each new application is treated as an independent new project, with resources assigned as available.	New applications are subject to review and approval by an architectural review board.	An architectural review board is empowered to mandate project changes in the interest of maximizing cross-enterprise benefits.	The disciplined enterprise focus has resulted in significant service reuse, faster time-to-market, and less custom development.	New application development is minimal, being replaced by rapid application assembly resulting from a well-stocked catalog of services.
Skills and Methodologies in Place	Which statement best describes the skills and methodologies you have in place?	We have not yet performed a formal assessment of our teams' SOA skills and methodologies.	We have assessed our skillsets, performed a gap analysis, and begun training for key SOA technologies.	We have established a baseline for skills and methods, and it is standardized across the enterprise.	We can certify and measure that the required skill levels and methodologies are in place across the organization.	With a basis in metrics, we formally invest in new skills and methods to remain ahead of the market.
Modeling and Abstraction Techniques	To what extent are modeling and abstraction techniques being employed in your environment?	Limited and inconsistent across projects, and confined to IT.	Under development and undergoing standardization, but still confined to IT.	Being employed in a consistent manner across all new projects, and beginning to involve business analysts.	Being expanded to include legacy systems as well as new development.	Model lifecycle begins and ends with business analysts.
INFORMATION	**Question**					
Data Standards and Canonical Formats	How much work has been done in the area of data standards and canonical formats?	No data standards or canonical data formats are in place.	Design has begun on data representation standards and canonical formats for key business documents.	Enterprise data model is under construction for key business data.	Initial plan for data standardization is complete and fully deployed.	Plan continuously evolves to address industry requirements from all our key customers and partners.
Metadata Management	To what extent do you practice metadata management as part of the IT process?	Metadata management is not viewed as important, and thus is not part of current plans or designs.	Design has begun on a metadata management solution to address specific business areas.	Plan is being implemented for specific types of business data.	Plan for metadata management is complete and fully deployed for all business data deemed critical to our core processes.	Our data management plan can be regularly measured, tuned, tested, and redeployed as conditions change.
Single Source of Truth	How would you describe your organization's ability to provide a single source of truth for key data?	There is no common data model in place for any application.	A common data model has been constructed but is in limited use by only one or two applications.	Multiple applications are leveraging a common data model and sharing data access logic.	Our most critical applications are tied into the common data model, but data duplication and cleansing are still an issue.	We have reliable data hubs that provide a single source of truth while also cleansing and updating source systems.

Focus Area		1	2	3	4	5
PROCESS	**Question**					
Process Automation	How would you characterize your organization's effectiveness in automating key business processes?	Process automation is difficult, and we tend to automate processes as a tactical response as opposed to a strategic plan.	We are becoming more proficient with process modeling techniques, but the models are difficult to impose on our systems, which are still brittle.	Our business process modelers are aligning more closely with our integration developers, and we are enjoying success as more shared services become available.	Process automation techniques have become standardized in our environment, and the benefits of agility are being noticed across most lines of business.	Our automation and instrumentation capabilities have advanced to the point where processes employ self-diagnostics and tuning for optimum efficiency.
Composite Application Development	How evolved is your ability to discover and assemble services into a composite application?	Limited and inconsistent across projects, and confined to IT.	Growing capabilities for composite application development as the repository of shared services begins to grow.	Repeatable success being achieved with specific processes and lines of business.	Repeatable success across all lines of business as the service catalog grows and governance strengthens.	Application development has transformed into application assembly, leading to rapid time to market and ideal levels of flexibility.
Process Measurement and Scoring	To what extent do you practice process measurement and efficiency scoring as part of the IT process?	Process metrics are either loosely defined or not communicated to IT.	Business and IT have only just begun to establish metrics for key processes.	We have establish meaningful metrics for business and IT, but only for a limited set of processes.	We have establish meaningful metrics for business and IT across all key business processes.	Process metrics are highly evolved and serve as key indicators for IT and business alike.
ORGANIZATION	**Question**					
IT Alignment with Business Strategy	How would you describe your IT group's ability to align with business strategy?	Our IT group has infrequent contact with the business.	IT and business have begun holding planning meetings to discuss the impact of services on the business.	IT and business closely collaborate on service modeling to support key business processes.	IT understands business processes extremely well and is agile enough to supply quick resolution to most new business problems.	We rely heavily upon IT not only for the planning and execution of critical business strategy, but for innovation and differentiation in the market.
Change Management	What steps have you taken to prepare people in IT and business for the notion of shared services?	The impact of SOA is not well understood, and so we spend little time and effort planning for its eventual adoption.	We are just beginning to learn from experience how SOA will change the way we work.	Best practices are being discovered and published throughout the organization.	Best practices are being converted into policies where appropriate, and we can track and model the impact they are having on the business.	We possess a clear understanding of SOA's value to our business and have formalized a change management strategy for driving it thoughout the enterprise.
Business Involvement and Understanding	To what extent is the business involved and aware of the solutions being enabled by shared services?	The few services that exist are confined to a limited audience within IT only.	The availability and benefit of reusable services is well understood by IT, but business people are just beginning to understand their potential impact.	To gain greater understanding and commitment from the business units, we have begun to capture and model the business and IT benefits we expect to accrue from our reusable service layer.	Using our model of expected benefits as a guide, we are able to monitor and report on the tangible and intangible benefits that flow from shared service layer.	Senior management has a clear understanding of the costs and benefits of SOA, and they are fully committed to evolving the shared service model across all levels of business.

Focus Area **GOVERNANCE**	**Question**	**1**	**2**	**3**	**4**	**5**
Funding and Accounting	Which scenario best describes your organization's funding and chargeback model?	Each business unit treats services as a standalone cost because there is no service reuse in our organization.	To drive efficiency and reuse of shared services, explicit funding is provided in the IT budget for a centralized architecture team.	Service reuse is occurring across the enterprise, but no formal cost allocation model has been addressed yet.	We have developed a budget complete with service usage metrics that allow us to implement an effective chargeback model.	Careful metering of shared service usage allows us to develop effective models for capacity planning, risk avoidance, and budget distribution
Cross-Organizational Involvement	How wide is the scope of organized SOA efforts within your organization?	Confined to limited groups of developers in IT.	Service awareness is pervasive throughout IT.	Service awareness spreading to business process owners in certain lines of business.	Services becomes a formal strategy addressed by cross-organizational review boards.	The services strategy is understood and mandated across all lines of business.
Policies, Reporting, and Exception Handling	How are policies created, enforced, and reported on within your organization?	They are treated as additional work for IT and are handled on an ad-hoc basis.	Roles and responsibilities have been formally modeled to address policy management and reporting.	Individual business units have centralized their policy enforcement, reporting, and exception handling.	A single centralized team has purview over policy, compliance, and exception handling.	A central policy team can dynamically implement rule and policy changes independent of underlying IT systems.

Bibliography

[Alonso 2004] G. Alonso, F. Kasati, H. Kuno, V. Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer, 1st edition, 2004.

[Bennett 1999] K.H. Bennett, M. Ramage, M. Munro. *Decision Model for Legacy Systems*. IEE Proceedings Software, IET, 1999.

[Bergey 2001] J. Bergey, L. O'Brian, D Smith. *Options Analysis for Reengineering (OAR): A Method for Mining Legacy Assets*. CMU/SEI-2001-TN-013, CMU Software Engineering Institute, 2001.

[Bianchi 2003] A. Bianchi, D. Caivano, V. Marengo, G. Visaggio. *Iterative Reengineering of Legacy Functions*. IEEE Transactions on Software Engineering, 2003, IEEE, 2003.

[Bisbal 1999] J. Bisbal, D. Lawless, B. Wu, J. Grimson. *Legacy Information Systems: Issues and Directions*. , IEEE Software, 1999.

[Bosworth 1994] M.T. Bosworth. *Solution Selling: Creating Buyers in Difficult Selling Markets*. McGraw-Hill, Inc., 1st edition, 1994.

[Britton 2004] C. Britton, P. Bye. *IT Architectures and Middleware: Strategies for Building Large, Integrated Systems*. Addison-Wesley, 2nd edition, 2004.

[Brooke 2001] C. Brooke, M. Ramage. *Organizational scenarios and legacy systems*. Internation Journal of Information Management, Elsevier Science Ltd., .

[Brown 2002] A. Brown, S. Johnston, K. Kelly. *Using Service-Oriented Architecture and Component-Based Development to Build Web Service Applications*. , Rational Software Corporation, 2002.

[CBDI SOA] D. Sprott. *CBDi Forum Report: The SOA Maturity Model*. 2005

[Cherbakov 2005] L. Cherbakov, G. Galambos, R. Harishankar, S. Kalyana, G. Rackham. *Impact of service orientation atthe business level*. IBM Systems Journal, IBM, 2005.

[Chikofsky] E.J. Chikofsky, J.H. Cross II. *Reverse Engineering and Design Recovery: A Taxonomy*. , IEEE Software, 1990.

[CMMI] unknown. *CMMI - Capability Maturity Model Integration*. 2008, http://www.sei.cmu.edu/cmmi/ (last accessed 26-05-2008)

[Cormella-Dorda 2000] S. Comella-Dorda, K. Wallnau, R.C. Seacord, J. Robert. *A Survey of Legacy System Modernization Approaches*. CMU/SEI-2000-TN-003, CMU Software Engineering Institute, 2000.

[de Lucia 2001] A. de Lucia, A.R. Fasolino, E. Pompella. *A Decisional Framework for Legacy System Management*. Proceedings of the International Conference on Software Maintenance (ICSM 2001), IEEE, 2001.

[Elfatatry 2007] A. Elfatatry. *Dealing With Change: Components Versus Services*. Communications of the ACM, ACM, 2007.

[Eurotransplant] unknown. *Eurotransplant homepage*. 2007, http://www.eurotransplant.nl/ (last accessed last accessed 06-05-2008)

[Fenton 1997] N. E. Fenton, S. L. Pfleger. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing, 2nd edition, 1997.

[Gartner 2002] D. W. McCoy. *Business Activity Monitoring: Calm Before the Storm*. , Gartner Inc., 2002.

[Gartner 2007] M. Driver. *How to Maneuver Oracle Forms Into an Ideal Position for Next-Generation Challenges*. Gartner Inc., 2007

[Hammer 1993] M. Hammer, J Champy. *Reengineering the Corporation: A Manifesto for Business Revolution*. Harper Business, edition, 1993.

[Henderson 1993] J.C. Henderson, N. Venkatraman. *Strategic Alignment: Leveraging Information Technology for Transforming Organizations*. IBM Systems Journal, Vol 38, IBM, 1993.

[Hevner 2004] A. R. Hevner, S. T. March, J. Park, S. Ram. *Design Science in Information Systems Research*. MIS Quarterly, , 2004.

[IBM SOA] R. High, S. Kinder, S. Graham. *IBM's SOA Foundation: An Architectural Introduction and Overview*. 2005

[InfoWorld 2007] Krill, P.. *Industry report: SOA is overly hyped*. 2007, http://www.infoworld.com/article/07/08/20/soa-report_1.html (last accessed)

[ISO/IEC 2001] unkown. *ISO/IEC 9126-1:2001 - Software engineering -- Product quality*. 2001, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=22749 (last accessed)

[Jha 2005] M Jha, P. Maheshwari. *Reusing Code for Modernization of Legacy Systems*. Proceedings of the 13th IEEE International Workshop on Software Technology and Engineering Practice (STEP'05), IEEE, 2005.

[Kontogiannis 2007] K. Kontogiannis, G. A., Lewis, D. B. Smith, M. Litoiu, H. Müller, S. Schuster, E. Stroulia. *The Landscape of Service-Oriented Systems: A Research Perspective*. International Workshop on Systems Development in SOA Environments (SDSOA'07), IEEE, 2007.

[Lewis 2005] G. Lewis, E. Morris, L O'Brien, D. Smith, L. Wrage. *SMART: The Service-Oriented Migration and Reuse Technique*. CMU/SEI-2005-TN-029, , 2005.

Lewis 2006: G. Lewis, E. Morris, D. Smith, Analyzing the Reuse Potential of Migrating Legacy Components to a Service-Oriented Architecture, 2006

[OASIS SOA] D. Nickull. *OASIS SOA Reference Model TC*. 2008, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm#technical (last accessed 26-05-2008)

[OG OSIMM] unknown. *OSIM: The Open Group Service Integration Maturity Model*. 2007, http://www.opengroup.org/projects/osimm/ (last accessed 26-05-2008)

[OMG 2006] unknown. *BPMN 1.0 specification*. 2006, http://www.bpmn.org/ (last accessed)

[OPENGROUP 2006] Harding, C.. *Definition of SOA*. 2006, http://opengroup.org/projects/soa/doc.tpl?gdid=10632 (last accessed last accessed August 23rd, 2007)

[ORACLE 2005] unknown. *Oracle Forms - Oracle Reports - Oracle DesignerStatement of Direction - September 2005*. 2005, http://www.oracle.com/technology/products/forms/pdf/10g/ToolsSOD.pdf (last accessed)

[Oracle 2007] unknown. *Oracle Unified Method 4.5*. 2007

[Oracle OUM] unknown. *Oracle Unified Method 4.5*. 2006

[ORACLE SOA] M. Afshar. *SOA Governance: Framework and Best Practices*. 2007

[Papazoglou 2005] M. Papazoglou, W.-J. van den Heuvel. *Service Oriented Architectures: Approaches, Technologies and Research Issues*. VLDB Journal, INFOLAB Tilburg University, 2005.

[Papazoglou 2006] M.P. Papazoglou, P.M.A. Ribbers. *e-Business: Organizational and Technical Foundations*. John Wiley & Sons, 1st edition, 2006.

[Papazoglou 2006, p. 468] Papazoglou, M.P., Ribbers, P.M.A.. *e-Business: Organizational and Technical Foundations*. John Wiley & Sons, 1st edition, 2006.

[Schwaber] Ken Schwaber. *SCRUM Development Process*. , http://jeffsutherland.com/oopsla/schwapub.pdf (last accessed last accessed May 1st, 2008)

[SERC 2005] unknown. *QUINT2 - The Extended ISO Model of Software Quality*. 2005, http://www.serc.nl/quint-book/ (last accessed)

[Sneed 2001] H. M. Sneed. *Recycling software components extracted from legacy programs*. Proceedings of the 4th International Workshop on Principles of Software Evolution, ACM, 2001.

[SQA] unknown. *ISO9126 - Software Quality Characteristics*. unknown, http://www.sqa.net/iso9126.html (last accessed last accessed 12-03-2008)

[Verdugo 1988] G. Verdugo. *Portfolio Analysis - Managing Software as an Asset*. Proceedings of InternationalConference Software Maintenance Management, Software Maintenance assoc., 1988.

[Ward 2003] J. L. Ward, J. Peppard. *Strategic Planning for Information Systems*. John Wiley & Sons, 3rd edition, 2003.

[webMethods 2005] unknown. *The Business Case for SOA*. 2005, http://www1.webmethods.com/PDF/The_Business_Case_for_SOA.pdf (last accessed)

[West 2006] M. West, B. Guptill, M. Koenig. *SOA Reality Check: Three Waves of Adoption through 2012*. Saugatuck Technology Inc., 2006

[WP 07a] unknown. *Business Process Modeling Notation*. 2007, http://en.wikipedia.org/wiki/Business_Process_Modeling_Notation (last accessed last accessd 22-11-2007)

[Zhang 2004] Z. Zhang, H. Yang. *Incubating Services in Legacy Systems for Architectural Migration*. Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04), IEEE, 2004.