1 Introducing Agile Service Development

M.M. Lankhorst, W.P.M. Janssen, H.A. Proper, M.W.A. Steen

This chapter introduces the topic of our book: Agile service development. It describes the economic importance of services, defines the service concept and shows how it may provide a handle on several management issues. Furthermore, it introduces the notion of agility, applied at organizations, at their service development processes, and at the services themselves. Finally, this chapter positions our work in the context of enterprise engineering and explains what its core contributions are.

1.1 Introduction

Economies around the globe have evolved to become *service economies*. This is in particular the case for Europe and the USA, but emerging markets are expanding in this direction as well. Service sectors are responsible for about 73% of the GDP in Europe (CIA 2011), 55% in India and 43% in China. As the Europe 2020 Strategy (European Commission 2010) makes clear, Europe's future wealth and well-being of its citizens depend on how effectively its businesses can innovate and respond to changing markets, technologies and consumer preferences. We therefore need a better understanding of how innovation is changing and how the traditional divide between manufacturing and services is blurring.

Organizations (including companies, government agencies, et cetera) that were traditionally production-oriented are now also adopting new service-focused business models. Consumers no longer just want a printer or a car. They rather ask for a *printing service* or a *mobility service*. Many types of services are provided (initiated and/or delivered) through the Internet. As a result, services-oriented organizations increasingly exploit new devices, technologies and infrastructures, such as smartphones, tablets or interactive televisions, to improve their customers' experiences. Innovation is no longer the preserve of research and development laboratories but has become more of a distributed, cultural phenomenon, where the processes for developing new goods and services, channels to market and revenue models are evolving in response to new technical opportunities, increased customer engagement in innovation, and changing organizational structures.

Despite its importance, the level of professionalism in developing services cannot match the level of expertise in product development. Business cases, user studies, design alternatives and actual development are not really linked, and information and knowledge is lost *en route*. Especially in the case of IT-based services, initial requirements are underspecified, leading to change requests in the process, with higher cost, longer time to market, and increased risk of disappointing customers. While this shift towards a service economy happened, organizations have also seen their pace of change accelerate steadily. This correlates with the increasing speed of development in IT, exemplified by Moore's law and the rapid rise of the Internet and mobile Internet, which in turn have driven customer demands and expectations. Organizations need to deal with this and adapt their way of working to increase their capabilities in anticipating and responding to such developments. *Agility* is the ability to deal easily with such changing requirements and environments. Agile ways of working embrace change as a positive force and harnesses it for the organization's competitive advantage.

At the same time, many organizations find themselves bogged down by a legacy of large, inert, complex systems and business processes. Often, traditional 'waterfall' development processes have been used to create these systems. These systems, and their development processes, cannot cope with the speed of change required by the modern day environment of the organization. Moreover, traditional software development projects have a dismally low success rate, due to both poor project planning and poor execution. The often quoted Chaos reports (Standish Group 2010) are a well-known source of this observation, but there are many more indicators. For example, there is a strong correlation between project size and failure rate (Verhoef 2002). In general, we struggle to successfully manage large IT projects and ensure they finish on time and within budget.

So-called agile methods for software development, such as Scrum and Extreme Programming, have become very popular in the software engineering community; according to a recent study by Forrester, 35% of the organizations surveyed already have mature agile methods in place and 33% were implementing agile (Forrester 2009). This popularity is not only because these methods feel less of a straitjacket to engineers but also because they are of help in realizing software systems that are better aligned with business and user needs, with a smaller risk of cost and time overruns.

Another way of managing large scale IT-related projects and programmes is through the use of architecture (Zachman 1987, The Open Group 2011). This used to be the domain of IT experts only, but nowadays includes the design of an enterprise as a whole. This is for example reflected by a definition of enterprise architecture as submitted to a survey of The Open Group by the Enterprise Architecture Research Forum (2010): '*The continuous practice of describing the essential elements of a socio-technical organization, their relationships to each other and to the environment, in order to understand complexity and manage change.*' Empirical support for the business value of architecture has also been shown, for example in the work of Slot (2010).

Given the observation that agile methods and enterprise architecture both can contribute to effective agility, it seems logical that agile methods and architecture should be combined. Until now, however, this has been a somewhat awkward marriage. Many agile practitioners tend to see architecture as 'Big Design Up Front', a big no-no in agile development. However, as we will describe later, agile and architectural approaches can be combined in a fruitful manner.

1.2 Services and Service Thinking

As we have outlined in the introduction of this chapter, service sectors have become a mainstay of the economy. But the service concept also serves an important purpose in business and IT management. This encompasses different aspects, ranging from determining the strategic orientation of the organization and its IT, to management and control of delivery and operations. Furthermore, the information systems landscape itself, especially of large, information-intensive organizations, has become a complex field that combines all kinds of concepts, paradigms, building blocks, and instruments. How can we get a grip on this multifaceted landscape?

It is impossible to manage all these different elements individually. Some of these are too fine-grained, such as business rules or events; some are too IT-centric, such as business objects or components; some are too large and serve too many purposes to manage them as a single functional element, such as complete business applications like ERP systems; and some of these, such as business processes, are too business-specific to provide a management handle on more generic IT functionality. We need a concept that is in between these other notions and captures the essence of what an organization does or means for its surroundings: *service*.

In short, a service is a piece of functionality that offers value to its environment. By concentrating on service development, we focus on the value that organizations provide to their environment (customers, citizens, society). Of course, these services are realized by all kinds of business processes, software applications and technical infrastructure. However, these are subordinate to the services they deliver. Traditionally, agile methods are strongly focused on software development; here, we take a much broader scope, applying agile principles and practices to more than just software.

Using the notion of service as core concept in guiding the development of organizations, both for business and for IT design, has several advantages. First, services provide a clean separation of the 'what' and 'how'. A service provides a clear interface to its functionality, without disclosing how this functionality is realized internally. As such, a service is self-contained and has a clear purpose and function from the perspective of its environment. Its internal behaviour, on the other hand, represents what is required to realize this functionality. For the 'con-

sumers' or users of a service, the internal behaviour of a system or organization is usually irrelevant: they are only interested in the functionality and quality that will be provided.

In this way, services also facilitate interoperability, minimizing the necessary shared understanding: a service description and a protocol of collaboration and negotiation are the primary requirements for shared understanding between a provider and user of some service. Therefore, services may be used by parties different from the ones originally conceived, or used by invoking processes at various aggregation levels.

This also points to the second advantage of the service notion: a service is independently useful and therefore has a manageable level of granularity. Since it delivers a concrete business contribution, it is the subject of service-level agreements, its performance can be monitored separately, it can be combined with other services to provide new functionality, while its delivery can be bought from and sold to other organizations.

Finally, the service concept provides a potential bridge between business and IT vocabulary. In business terms, 'service' signifies what the organization does for its customers; more recently, IT has started to use the word 'service' for concrete, independent units of business functionality delivered via a software interface. Both uses of the word are based on the concrete contribution to the environment and the relatively self-contained character of a service.

This is of course not really new. Organizations have long been thinking in terms of the services provided to customers, and internal business processes are designed to provide these services. Software engineers think in terms of functional interfaces, information hiding and encapsulation. Service thinking, however, can also be applied to, for example, internal business processes and software applications, rendering them into 'service networks': services become the core building block of the entire information ecosystem.

Service orientation also stimulates new ways of thinking. Traditionally, applications are considered to support a specific business process, which in turn realizes a specific business service. Service orientation also allows us to adopt a bottom-up strategy, where the business processes are just a mechanism of instantiating and commercially exploiting the lower-level services in a collective offering to the outside world. In this view, the most valuable assets are the capabilities to execute the lower-level services, and the business processes are merely a means of exploitation.

Hence, by concentrating on agile development of business and software services, we focus on the *value* that organizations provide to their environment. Of course, these services are realized by all kinds of business processes, software applications and technical infrastructure. However, these are subordinate to the services they deliver. Traditionally, agile methods are strongly focused on software development; here, we take a much broader scope, applying agile principles and practices to more than just software.

1.2.1 Service Definitions and Properties

Let us be more clear about what we mean by the elusive notion of 'service'. The service concept is widely used in economics, business science, innovation, business process engineering, and IT. However, the concept is used in several ways across these fields. An extensive number of interpretations from the literature has been reviewed by Quartel et al. (2007). They list the following types of definition for the term 'service':

- *Value creation*. In economics and business science, a service is seen as the non-material equivalent of a good, creating value for the service consumer, for example by (Quinn et al. 1987).
- *Exchange*. Many definitions focus on the exchange between the provider and consumer of a service, such as the definitions of Spohrer et al. (2007) and Papazoglou and Heuvel (2007).
- *Capability*. Often the service concept is defined as an abstract resource that represents some capability, for example by the W3C (2004) and the OASIS SOA Reference Architecture (OASIS 2006, OASIS 2011).
- *Application*. Web services, but also services in general, are commonly seen as applications (pieces of software) that can be accessed over the Web, for example in (W3C 2004).
- *Observable behaviour*. In data communication, a service is traditionally defined as the observable, or external, behaviour of a system, for example by Vissers et al. (1986).
- *Operation*. In object-oriented and component-based software design, each operation or method defined on an object or component is usually seen as a service of that object or component.
- *Feature*. In the telecommunications domain the term service is used to refer to a feature that can be provided on top of the basic telephony service, such as call forwarding, call back when busy and calling line identification.

Generalizing the definitions listed above, Quartel et al. (2007) identify four defining characteristics of services:

- **Services involve interaction**. A service involves one or more interactions between a *service user* and some system that provides the service, also called *service provider* or *service system*.
- **Services provide value**. The execution of a service provides some value to the user and the provider. In case of IT services, this value may only involve 'intangible benefits', such as the change in possession of goods and money. For services in general, the value may also involve 'tangible things', such as the actual exchange of parcels using a parcel delivery service.
- **Services define units of composition**. Services are units of composition. Business processes and supporting applications are composed from services, which

define smaller business process or application pieces that may be reused when chosen properly.
- **Services are a broad spectrum concept**. The service concept is meant to be applied at successive abstraction levels along a broad spectrum of the design process, i.e., from specification to implementation.

1.2.2 Our Definition of Service

The definitions and characteristics above lead us to our own definition of service, which aims to be both concise and generally applicable to different kinds of services.

> A *service* is a unit of functionality that a system exposes to its environment, while hiding internal operations, which provides a certain value (monetary or otherwise).

This definition, reused from the ArchiMate 2.0 standard (The Open Group 2012), is generic enough to encompass most of the business-oriented definitions above. It focuses on the functionality and value inherent in a service and stresses that a service should hide its internal operations, i.e., its users should perceive it as an integrated whole that can be used on its own. The definition does not specifically speak about a 'consumer' or 'provider' of a service, unlike some of the definitions reviewed above. Although a service must of course be provided and consumed, we do not wish to limit ourselves a priori to an implied one-on-one relation between a provider and a consumer, since services may be used and produced by more complex groups, networks or other structures of actors. Furthermore, the same service may be offered by different parties, and we do not want to suggest that the service is tied to a specific provider. Hence, our definition simply speaks of a service establishing value to its environment.

A service represents only the 'externally visible' behaviour of a 'service system' (see below), as it is experienced by the users of the service. A service should not be confused with the interface or channel at which clients can obtain that service; for example, an organization may offer the same information service via its website, call centre, or front desk. Also, a service offer need not be targeted at a pre-existing, specific demand; it may also be used to *create* such a demand: 'if you build it, they will come', as the famous movie quote from Field of Dreams put it. Hence, the value established by a service may only become clear *after* it has been created and used.

We can distinguish different types of services:

- A *business service* is a service that is provided by an organization to its environment, or by an organizational unit to the organization.

- An *application service* is a service that is provided by a software component to its environment (both users and other software components).
- An *infrastructure service* is a service that is provided by some infrastructure element (e.g. a hardware device) to its environment.

These services may be viewed as ordered in layers that support each other: business services may be delivered in part or completely by way of software and infrastructure services. All these services are realized by service systems, which in turn may rely upon other services.

> A *service system* is a value-coproduction configuration of people, technology, other internal and external service systems, and shared information (such as language, processes, metrics, prices, policies, and laws).

This recursive definition, taken from (Spohrer et al. 2007), highlights the fact that service systems have an internal structure and may be part of an external service network. A single application may be a service system, realizing a specific software service; individuals and organizations are service systems, and at the extreme end of the spectrum, so are entire nations and economies.

Even though we define the general concept of service as a self-contained unit of functionality that establishes a meaningful value to its environment, it sometimes is necessary to be more specific about the fact whether we refer to a service consumer or producer, or to a service request or offering, or to a service delivery as a whole. In particular the following concepts are useful:

> *Service delivery*: The combination of a service offering, execution and completion as conducted by the service producer.
>
> *Service consumption*: The combination of a service request to a service producer, and the associated acceptance of its completion.

These concepts also resonate well with the generic transaction pattern as for instance described by Dietz (2006) and with the definition of service as suggested by Albani, et al. (2009) and elaborated by Terlouw (2011).

1.2.3 Service Development as a Wicked Problem

The development of new services is likely to take place in situations where technology platforms evolve rapidly, introducing several technological uncertainties, while at the same time several stakeholders with conflicting stakes are involved. This confronts service designers with major challenges. To add more spice to the challenges, it may not even be clear what the business model will be for a new

service. Competitors struggle with the same challenges and potential benefits, hence doing nothing is not an option.

As Hevner et al. pointed out, this type of design problems is 'wicked', i.e., no optimal solution can be found in reasonable time (Hevner et al. 2004, p. 89):

> Given the wicked nature of many information system design problems, however, it may not be possible to determine, let alone explicitly describe, the relevant means, ends, or laws. Even when it is possible to do so, the sheer size and complexity of the solution space will often render the problem computationally infeasible [...] In such situations, the search is for satisfactory solutions, i.e., satisficing (Simon 1996), without explicitly specifying all possible solutions. The design task involves the creation, utilization, and assessment of heuristic search strategies. That is, constructing an artifact that 'works' well for the specified class of problems.

The concept of 'wicked problem' was first coined by Rittel and Webber (1973). They characterize this wickedness as follows:

1. You don't understand the problem until you have developed a solution.
2. Solutions to wicked problems are not right or wrong.
3. Every wicked problem is essentially unique and novel.
4. Wicked problems have no stopping rule.
5. Every solution to a wicked problem is a one-shot operation.
6. Wicked problems have no given alternative solutions.

Jeff Conklin (2005) complements the notion of wickedness with the concept of social complexity, stating that:

> Social complexity means that a project team works in a social network, a network of controllers and influencers including individual stakeholders, other project teams, and other organizations. These relationships, whether they are with direct stakeholders or those more peripherally involved, must be included in the project. For it is not whether the project team comes up with the right answer, but whose buy-in they have that really matters. To put it more starkly, without being included in the thinking and decision-making process, members of the social network may seek to undermine or even sabotage the project if their needs are not considered. Social complexity can be can be understood and used effectively, but it can be ignored only at great peril.

Social complexity exacerbates a problem's wickedness. In terms of Conklin: 'Fragmentation = wickedness × social complexity'. For such wicked and socially complex problems, top-down, waterfall-style design approaches fail. This requires us to look for different ways of thinking and working.

Moreover, as Ciborra (1992) argued, 'bricolage', emergence and local improvisation, instead of central control and top-down design, may lead to strategic advantages: the bottom-up evolution of socio-technical systems will lead to something that is deeply rooted in an enterprise's organizational culture, and hence much more difficult to imitate by others. Such bottom-up tinkering may also lead to much quicker responses to a changing environment than a highly structured and formalized design process; this speed itself may be a strategic advantage over competitors, and as we have argued before, the increasing speed of change in the environment requires organizations to be ever more responsive.

A similar line of reasoning is followed in the design thinking approach, as introduced by Rowe (1987) and made popular by Brown and Kelley at the design company IDEO (Brown, 2009). Design thinking emphasizes the role of iterative design and strong user involvement to tackle the social complexity of many design problems. Iterative design here involves early prototyping and user feedback, not only for objects to be designed, but also for services. Design thinking is not so much process oriented. But distinguishes three overlapping design spaces: inspiration, ideation and implementation. In these spaces desirability, viability and feasibility of a service or product are balanced.

1.2.4 The Need for Agility

Wickedness and social complexity are not only a challenge to service development. They are, for example, a challenge to software development as well. In the context of software development processes this has, over the last decade, given rise to the notion of 'agility', with popular software development methods such as Extreme Programming and Scrum, and with the well-known Agile Manifesto (Beck et al. 2001) as a kickstarter. Agile methods, with short iterations, close customer contact, continuous adaptation, self-organization and cross-functional teams, have been adopted by an increasing number of organizations.

In battling wickedness and social complexity in the context of service development, we look at agility as a means to deal with the complexity and dynamicity. However, the traditional agile approaches only concern the agility within the development process. The object of that development, a service system, comprising both IT and business elements, should itself also be flexible and adaptable, to accommodate future changes. This is where we see an important role for architecture: designing service systems in such a way that they are flexible in the areas that may undergo rapid changes, and on the other hand offer a stable infrastructure for these services. This may seem paradoxical: agility and flexibility often arise from the use of a set of standardized 'building blocks' and interfaces. Lego is a good example: you can build almost anything from these standardized blocks with their fixed studs. Agile architectures also consist of stable elements that are easily configured and combined. We therefore address different kinds of agility:

Business agility: using change as an essential part of your enterprise strategy, outmaneuvering competitors with shorter time-to-market, smarter partnering strategies, lower development costs and higher customer satisfaction.

Process agility: using agile practices for design and development, focused on people, rapid value delivery and responsiveness to change.

System agility: having organizational and technical systems that are easy to reconfigure, adapt and extend when the need arises.

These different types of agility reinforce each other: if an enterprise's infrastructure, applications or business processes are more flexible, an iterative and incremental development process can more quickly and easily add value, and strategy execution is facilitated. Thus, these three kinds of agility are the foundation for the agile enterprise.

The core of all three kinds is that *uncertainty* is given an explicit and prominent place. Whereas traditional management, design processes and architectures plan for fixed goals and situations, agile methods and systems are aware of the uncertainties of their environment and know that they are aiming at a moving and often ill-defined target. Later on in this book, we will see how we give this uncertainty an explicit place in our way of working and in the artefacts we design.

The notion of agile systems also leads us to define agile services, based on our previous definition of the service notion:

> An *agile service* is a service (i.e., a self-contained unit of functionality that establishes a meaningful value to its environment) that has the ability to accommodate expected or unexpected changes rapidly.

The definitions of business, application, and infrastructure services can be augmented likewise.

An integrated approach for agile methods, architectures and services, based on sound engineering principles, is not yet available. Some organizations have practical experiences with elements of such a new way of working; others have only just embarked on such a trajectory or first want to gain more insight in its potential benefits and pitfalls. This book aims to fulfill that need.

1.3 Agile Enterprise Engineering

Management science and organizational science have long aimed to take a science-based approach to the design and evolution of enterprises. However, the complexities of modern day society where organizations, business, and IT 'fuse' to a complex whole, require a powerful instrument that enables effective and evidence-based decision making. In our view, now more than ever there is an evident need to complement the existing social sciences based views on the development of organizations with a model-oriented perspective on the design of enterprises, inherited from the engineering sciences. This will allow the creation of an evidence-based approach to the design, and associated decision making, of the complex, and open, socio-technical systems modern day enterprises are. Such a model-oriented and evidence-based approach will enable senior management to make better founded decisions, based on actual insight.

The core idea is to provide a model-based stream as *part of* change efforts, enabling evidence-based decision making on the future direction of the enterprise.

Models provide a good way of understanding where an enterprise is 'at', where it is currently moving 'towards', analyse the desirability of these, and articulate where it should ideally be moving 'towards'.

Complementing the development of organizations with a model-based engineering perspective, is comparable to the evolution of other engineering disciplines in the past, such as mechanical engineering, electrical engineering, or civil engineering. Initially, the intuition and experience of a craftsman was leading, but increasingly, this expertise was objectified and founded on scientific knowledge. Nowadays, all mature engineering disciplines are firmly rooted in the use of formal, mathematical models for predicting the various properties of their design artifacts in order to make the right decisions. The increased use of formalized business models, architecture models, risk assessment models, or valuation models also clearly points towards the increased use of a model-based approach to the design of enterprises.

Another development indicates the same process of maturation: the increased use of standards, not just on a technological level, but also in methods and techniques. IT management uses well-established frameworks such as COBIT (ITGI 2009), ITIL (ITIL 2011), and ASL (Pols & Backer 2007). Similarly, the increased popularity of architecture standards such as The Open Group Architecture Framework (TOGAF) (The Open Group 2011) and ArchiMate (The Open Group 2012) also demonstrates this maturation in the realm of enterprise architecture.

The Oxford English Dictionary (OED 2009) defines engineering as 'the branch of science and technology concerned with the design, building, and use of engines, machines, and structures.' This definition, especially the 'structures' part, also applies to (the structural parts of) enterprises and enterprise networks. Designing and operating business models, organizational hierarchies, work processes, information systems, and other parts of the various structures of enterprises, can be done with an engineering approach.

1.3.1 Limits to an Engineering Approach

While adding an engineering approach to the development of organizations makes sense, we should at the same time also recognize its limits. As already suggested above, we see an engineering approach as being complementary to existing approaches originating from management science and organizational science.

It would be a mistake to think that the use of formalized models and methods means that the design of organizations and their information systems becomes a deterministic exercise: drawing up plans and then faithfully executing them. The traditional engineering mindset presumes that there is a predefined problem worthy of a solution; however, in social and socio-technical systems such as the service systems we consider here, problems and solutions co-evolve in a closely connected way. The social stream in change is crucial in ensuring that the models of

the enterprise's design are indeed aligned what is actually established in the real social-technical system that makes up the enterprise. We should avoid using a 'blueprinting-only' (in terms of the change management 'colours' of De Caluwé and Vermaak (2008)) style of change management, i.e., not approaching organizational problems with a top-down blueprinting approach, while ignoring the softer, social and political aspects of organizations. A classical engineering approach to social systems may invite such a way of working, but social issues, for example cultural differences between partners in a merger, cannot be 'engineered' in a top-down, command-and-control like fashion. Furthermore, the rapidly changing environment of enterprises necessitates a flexible response, which cannot be provided by classical engineering methods only, as .An enterprise is first and foremost a *social* construct.

1.3.2 The Enterprise Engineering Manifesto

Taking an engineering approach to the design of enterprises is also one of the key points made by the *Enterprise Engineering (EE) Manifesto* (Dietz 2011). We regard this Manifesto as a laudable attempt to formulate the goal of evolving the development of enterprises into a proper engineering discipline. While we support the goals of the Manifesto, our discussions above on the agile and socio-technical aspects of the development enterprises do suggest several improvements.

While the manifesto justifiably focuses on enterprises as being essentially social-technical systems, its current wording suggests a rather traditional and linear view on the development of enterprises. Its first postulate states: 'In order to perform optimally and to implement changes successfully, enterprises must operate as a unified and integrated whole. Unity and integration can only be achieved through deliberate enterprise development (comprising design, engineering, and implementation) and governance.' This postulate presupposes that there is one optimum to strive for. However, different stakeholders are more than likely to differ in what they consider to be optimal; just take the different perspectives of a customer, shareholder or employee, for example. Agile methods explicitly take this multi-stakeholder view. They are aware of the wickedness and social complexity of the design problem at hand and try to find solutions that are sufficiently good from these different perspectives, instead of striving for an optimum, thus applying a satisficing approach (Simon 1996).

Nevertheless, in this context service development, with many different stakeholders and an uncertain and changeable environment, some level of guidance and control may be needed to keep local optimization and variation within bounds and to balance the needs of the various stakeholders. The Manifesto rightly emphasizes the role of architecture as an important instrument to provide such guidance. Architecture can serve a prominent role in explicitly designing for uncertainty: not

by rigidly planning for a predetermined future, but by providing mechanisms for adaptation in those places where future changes may be expected.

In conclusion, we think that the Enterprise Engineering paradigm provides an important step forward in the design and operation of organizations. Once again, however, it would be a mistake to think that the use of formalized models and methods means that the design of organizations and their information systems becomes a deterministic exercise: drawing up plans and then faithfully executing them. The traditional engineering mindset presumes that there is a predefined problem worthy of a solution; however, in social and socio-technical systems such as the service systems we consider here, problems and solutions co-evolve in a closely connected way. To address this complex and evolving web of relations and perspectives, we think that important lessons can be learned from the iterative and interactive ways of working of the agile movement. An Agile Enterprise Engineering Manifesto may therefore be in order.

1.4 Towards an Engineering Approach to Agile Service Development

A new perspective on service design processes is needed, providing development teams with the means to tailor their way of working to specific circumstances and deal with multiple stakeholder perspectives, bottom-up innovation and co-evolution of different service aspects. This book aims to provide steps in this direction. We advocate that agile development processes are much better suited to accommodate these needs than classical linear, top-down design processes, in which individual aspects are often developed separately and sequentially. The iterative character of agile processes, with a focus on people and interactions, close contact with customers and cross-functional teams that tackle different aspects of development at the same time, is a much better fit with the complex and multidimensional nature of service development.

Development processes should also be explicitly focused on observing changes in their environment and acting upon these. As we have argued before, the speed of change that organizations have to deal with keeps increasing, and processes must be responsive and even predictive in character to accommodate these changes. These properties should be designed into the development processes, which should be treated as systems in their own right. Such a systemic approach also requires the use of development processes that are self-aware, i.e., that use mechanisms and practices to observe their own performance and if necessary, change their own operation accordingly. This use of reflection is a common characteristic of agile methods. Scrum, for example, uses the ‘sprint retrospective’ meeting in which after each iteration, the way of working of the team is evaluated and adapted. In fact, this closed-loop, adaptive character is perhaps the most important

factor in the success of agile processes compared to the traditional open-loop, linear type of development.

This adaptive character of development processes does not mean that change knows no bounds. The complex nature of service design necessitates the use of sound engineering principles and techniques. External dependencies, technological complexity, regulatory compliance, risk management and other factors all require an approach of bounded or controlled variation. Architecture is a core discipline to provide such managed variation. It specifies the high-level, strategic or otherwise important principles and decisions that together span the design space, like a vector space in algebra.

Another important use of architecture is to explicitly design mechanisms in the operational processes and systems that support change. Not only should development processes be agile and adaptive, but the results they create should also be flexible and amenable to change. Various kinds of architecture and design models, ranging from domain, requirements and architecture models to detailed artefacts describing the inner workings of business processes and IT systems, play an important role in both controlling complexity and fostering change. Such models make business knowledge visible across the enterprise, which promotes coherence and consistency across the enterprise.

Moreover, a flexible infrastructure that can be configured with such models, instead of laboriously writing software code, may greatly enhance the agility of the organization and its systems for those specific aspects of agility that are captured by these models. Models can be changed more easily than code, and the effects of changes may be evaluated at the model level before processes and systems are changed, thus avoiding costly errors and re-implementations.

In agile development, the role of these models is not the same as in traditional design processes, however, where specialists each work on their own aspect models and then hand them over to the next person in the design chain. Rather, different models need to be evolved iteratively and in parallel, while guarding their mutual coherence and consistency. This is illustrated in Fig. 1.

This way of working with models has at least three important advantages:

1. Developing these models and other artefacts concurrently within a cross-functional team and in close cooperation with business stakeholders helps aligning the results with each other;
2. Using models and model-based views to discuss aspects of the service helps in aligning the result with stakeholder expectations in a very early stage, avoiding costly rework later;
3. Similarly, errors and misinterpretations can be detected early, by verification and testing at the model level, thus improving the quality and lowering the costs of the resulting services.

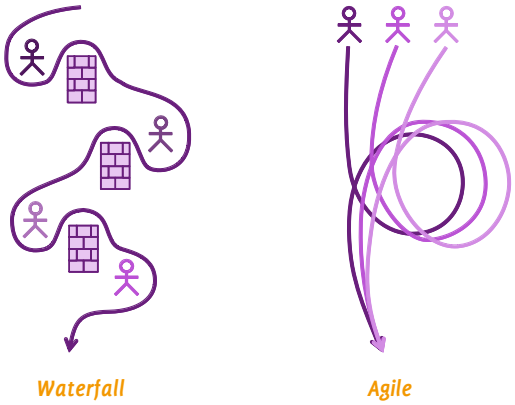

Fig. 1. Waterfall vs. agile process.

Next to this, the obvious advantages of iterative processes apply, such as early delivery of value and the possibility of changing course when circumstances change.

Our approach is different in another aspect as well: Whereas traditional development processes try to reduce uncertainty as early as possible, for example by having an extensive requirements engineering phase before starting the design, then writing complete functional specifications, technical designs, et cetera. we only reduce uncertainty when it is needed, but no sooner. And at that time we use information from sources that may offer certainty from *all* directions, not just the 'flow of the waterfall'. This information may for example come from decisions already taken on the business network of the service, models that have been worked out further, available building blocks, interface standards, available infrastructure elements, processes that are fixed because of regulatory compliance, and more. In this way, the collection of artefacts that jointly constitute the entire service, from abstract models of the value network to specific infrastructural components and detailed work instructions for employees, evolves as a whole, gradually and iteratively converging on the final result.

This approach requires that various models of service aspects can inform each other. To this end, we have defined a framework and set of basic concepts to which these models are mapped to capture their relationships. This will be discussed in Chap. 4 of this book.