Towards a Consistent Service Lifecycle Model in Service Governance

Michael Niemann
Department of Computer Science
Technische Universität Darmstadt, Germany
niemann@kom.tu-darmstadt.de

Michael Appel
Department of Computer Science
Technische Universität Darmstadt, Germany
mappel@rbg.informatik.tu-darmstadt.de

Nicolas Repp
Department of Computer Science
Technische Universität Darmstadt, Germany
repp@kom.tu-darmstadt.de

Ralf Steinmetz
Department of Computer Science
Technische Universität Darmstadt, Germany
steinmetz@kom.tu-darmstadt.de

ABSTRACT

Introducing an SOA in a company brings new challenges for the existing management. Small loosely coupled services allow the Enterprise Architecture to flexibly adapt to existing business processes that themselves depend on changing market environments. SOA, however, introduces a new implicit system complexity. Service Governance approaches address this issue by introducing management processes and techniques, and best practices to cope with the new heterogeneity. Service lifecycle management is one aspect of service governance. However, definitions and approaches vary greatly in this area. In this paper, we outline and compare existing service lifecycle approaches concerning their structure and defined phases. In particular, we challenge the purpose of the distinctions made between design time, runtime, and change time.

Keywords

Service lifecycle, Service-oriented Architecture (SOA), Service Governance, runtime, design time, change time

INTRODUCTION

During the past few years, the Service-oriented Architecture (SOA) paradigm has proved of value in enabling agile business processes due to the use of standards, loose coupling of services, and a dynamic service binding [Papazoglou 2003, Newcomer and Lomow 2004, Huhns and Singh 2005]. The introduction of an SOA, however, challenges the existing IT management. Small loosely coupled services allow the Enterprise Architecture to flexibly adapt to existing business processes that themselves depend on changing market environments. The classical IT Governance is defined to establish regulation of responsibility, authority, and communication to empower people in terms of decision rights [Weill and Ross 2004]. Furthermore, IT Governance is meant to establish mechanisms of measurement, policy standards, and control to enable the empowered people to carry out their responsibilities within the previously defined roles and authority [Clark 2005, Kjaer 2004]. However, these measures do not completely cover the new inbuilt complexity of an SOA. The number of new software artifacts dramatically increases, and the functional range of each service is rather small. These conditions motivate governance structures to regulate, direct, and control the SOA system, in particular targeting complexity reduction and conformity [Marks and Bell 2006, Niemann et al., 2008].

In these circumstances, lifecycle management is considered more crucial than in a governance approach for a monolithic IT system. In order to successfully operate a service-oriented system, services are subject to a multitude of regulations, e.g., service operation policies, technological policies, security policies, architectural policies, design time policies, and many more [Afshar 2007, Marks and Bell 2006, Mukhi and Plebani 2004, Niemann et al. 2008]. Governance regulations directly address service development and usage. Each service passes several phases such as specification and implementation before it is deployed, used, revised if necessary, and finally, replaced.

Many different approaches regarding service lifecycles have been developed and used by academia and software companies. Very often, one can find the distinction between *design time*, *runtime*, and *change time* – each of them covering a number of different lifecycle phases. In this paper, we outline and compare existing lifecycle approaches and, in particular, challenge the purpose of the distinctions made between these three ‘times’.

In the first part of this paper, we discuss related work concerning the service lifecycles. We compare existing approaches in order to identify characteristics and congruencies. In particular, we investigate and challenge the purpose of *change time* within the service lifecycle. Based on this discussion, in the second part of this paper, we present our service lifecycle approach that omits the change time aspect and, nevertheless, can describe all service lifecycles discussed in related work. We outline the ways in which each of the presented approaches can be mapped to our service lifecycle approach.

RELATED WORK

In general, lifecycle processes originate from the discipline of software engineering (SE). SE processes define any software's lifecycle from requirement analysis to implementation and maintenance. The main purpose is to ensure a structured software development process in a well-defined and cost effective way. [Sommerville 2006, Bhushan and Tayal 2007]

The typical lifecycles (the "waterfall model") consists of these phases: *requirement analysis and definition*, s*oftware design*, *implementation and testing*, *integration*, and *operation and maintenance* [Boehm 2007, Bhushan and Tayal 2007, Jacobson and Bylund 2000, Sommerville 2006]. It covers:

- *Phase 1 - Requirements analysis and definition* defines the software system's purpose, constraints and goals in cooperation with the system users. These aspects compose the system specification.
- *Phase 2 - Software design* distinguishes between hardware and software requirements and defines a general architecture. It covers the identification and description of software components and their relationships.
- *Phase 3 - Implementation and testing* comprises the realization of the defined design. It involves testing to ensure that the software meets the agreed specifications.
- *Phase 4 - Integration* includes system tests to ensure that interaction with other software components works faultlessly. Furthermore within this phase, the software is delivered to the customer.
- *Phase 5 - Operation and maintenance* usually is the longest phase. It includes software usage and monitoring, as well as error-correction processes. Should it become necessary to implement new requirements, a change process is triggered in this phase.

These phases are designed to form a loop. As soon as changes in functionality or requirements are registered, the phase *operation and maintenance* is interrupted and the lifecycle continues with the first phase, *requirements analysis*, or any other phase of the lifecycle, that can directly or better address the occurred problem [Sommerville 2006]. Figure 1 shows this "waterfall model".

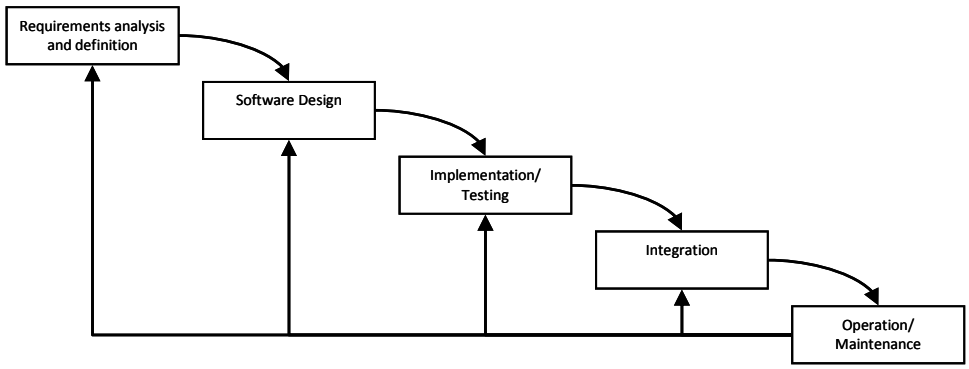

Figure 1. "Waterfall Model" [Sommerville 2006]

Services as part of an SOA are also software artifacts, however, their functionality scope is rather small and they appear in multiplicity. Obviously, however, slightly different conditions seem to be relevant when defining a lifecycle for services. Service lifecycle approaches developed are often divided into three super-phases: *design time*, *runtime*, and *change time* [Gu and Lago 2007, Systinet 2006, Matsumura 2007]. The lifecycle phases up to deployment are commonly referred to as *design time*. The usage phase is referred as *runtime*, followed by the *change time*, which addresses service change and revision.

Comparison of Existing Approaches

We conducted a mapping of ten service lifecycle propositions envisioned by industry and academia to the SE lifecycle. In the following, we introduce and discuss these approaches. We outline their structure, semantics, and particularities. In particular, we investigate whether and how the super-phases design time, runtime, and change time are addressed. We perform a matching of all presented lifecycles to the classic SE lifecycle model (cf. Figure 2). The white and grey lanes mark the phases corresponding to one of the SE lifecycle phases. State-intersection could not completely be avoided; however, it has been minimized. Dashed lines indicate the super-phases design time, runtime, and change time.

McBride at IBM defines four service lifecycle phases: *model, assemble, deploy,* and *manage* [McBride 2007]. *Model* covers the classical phases requirements analysis and design. *Assemble* refers to the phase 'implementation and testing'. *Deploy* maps to the phases 'integration' and 'implementation and testing', as it contains the testing of the assembled service and its further deployment. *Manage* is covered by the classical phase of operation; it refers to service monitoring and service compliance and not to service change. This approach describes the services' lifecycle at a rather high level. No detailed procedures and operations are defined. Furthermore, it does not include a system analysis, but grounds his lifecycle approach on IBM's SOA Governance lifecycle which consists of the very same phases.

Woolf at IBM proposes a five state service lifecycle model consisting of the phases *Planned, Test, Active, Deprecated,* and *Sunsetted* [Woolf 2006]. A service is planned, designed, implemented, and tested, where the phase *Planned* covers design and *Test* covers implementation and testing. Then it is set *active* to be used. During usage it can be marked deprecated to inform its consumers about a future retirement of the service. Finally, the service becomes *sunsetted*. Woolf's approach is, compared with McBride's lifecycle, a rather low level lifecycle. It differs from the classic SE lifecycle, as no cyclic iteration is present and an explicit integration or deployment is not considered. Woolf claims that service maintenance occurs very rarely. In spite of changing a service, according to Woolf, rather mostly *a new service* is designed and released. *Change*, therefore, is not often necessary. The classic SE lifecycle extends his approach by including the ability to reiterate the lifecycle.

The work by Strnadl [Strnadl 2007] proposes a lifecycle consisting of four development phases (*architecture, development, test,* and *production*) comprising twelve sub-phases. *Architecture* covers requirements analysis and design, while the phases *development* and *test* are specified separately. *Production* covers deployment and usage. The classic SE lifecycle extends this proposal by requirements analysis and by a service change process: Strnadl's lifecycle approach does not cover cyclic usage. Once developed and deployed, a service cannot be changed again.

Authors at Software AG (2008) present a non-cyclic lifecycle approach, which is rather detailed and similar to our approach. The authors consider the analysis of a service part of portfolio management. After the service is designed, built, tested, and deployed, it moves into the operation phase which covers service monitoring and change. No change time is taken into account in this lifecycle approach. A change phase as well as the ability to reiterate the lifecycle for service change is missing.

The approach by Windley (2006) explicitly distinguishes between design and runtime and corresponding phases such as *design, implement, deploy* at design time, and *manage*, and *retired* at runtime. Windley's approach does not include the possibility to prepare the service design by a requirements analysis, or a system analysis as defined by the SE lifecycle. As a cyclic approach, it allows a service to retire during the phase *manage*, or to be changed while reiterating the cycle and entering in the phase *design*. However, it is unclear exactly which processes are covered within the proposed *manage* phase – we assumed *operation*. Windley's approach does not (explicitly) consider change time.

Oracle's lifecycle [Wall 2006a, Wall 2006b] is a cyclic approach. A service traverses the *identifying business process* phase before being modeled. *Build and compose* obviously covers implementation and testing. This lifecycle is the only approach that implements a *publish and provision* phase before the service is integrated and deployed. "Securing" describes the process of service adaption to customer needs. A service is *secure*d *and manage*d after deployment (*integrate and deploy*), followed by the *evaluation* phase. *Evaluation* also covers service operation. A reiteration of the lifecycle is possible; the change process reenters in the first phase. Change time is not (explicitly) included within this approach.

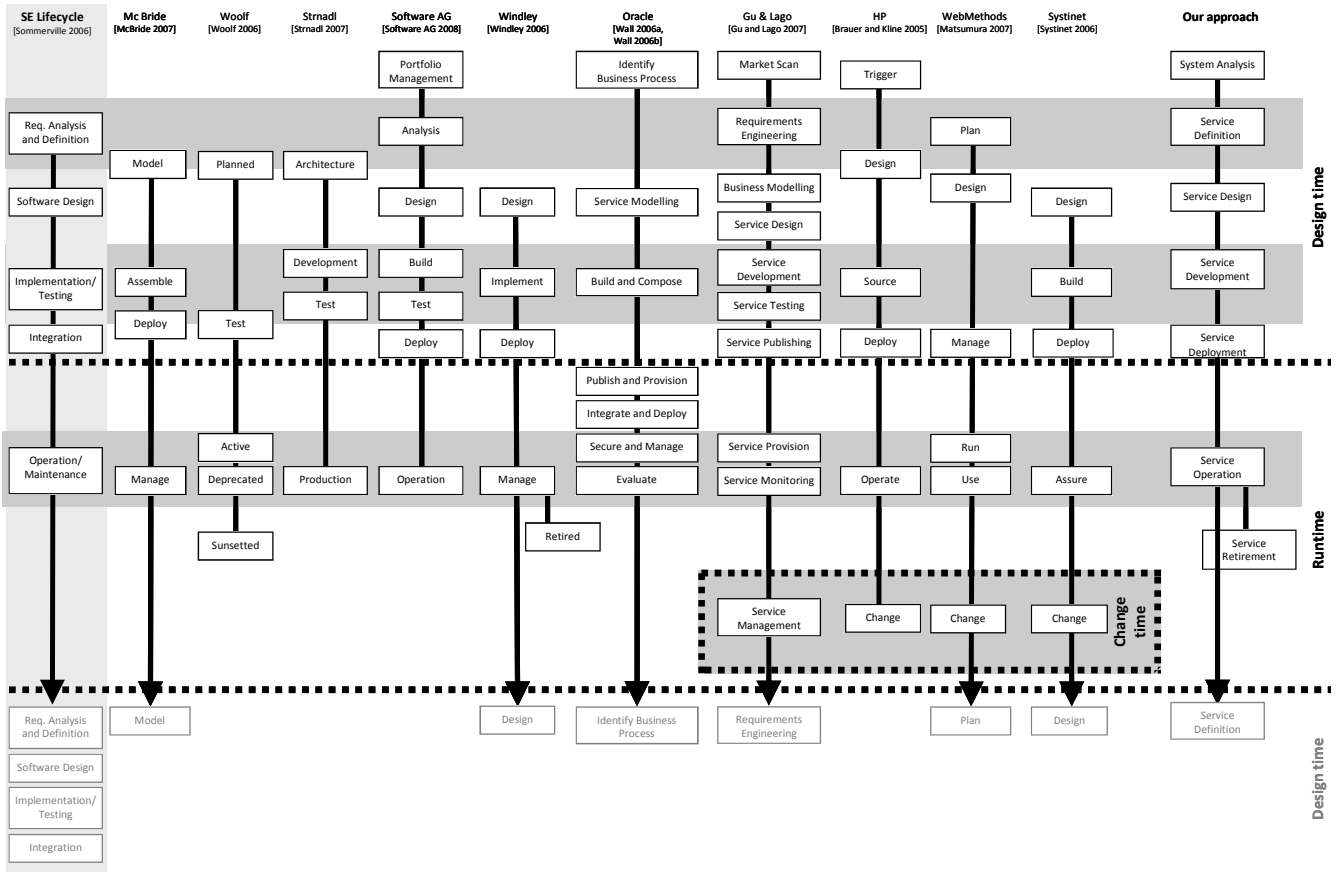

Figure 2. Service Lifecycle comparison

The lifecycle by Gu and Lago (2007) is a cyclic approach implementing several phases, starting with a *market scan* which maps to our system analysis phase (cf Fig. 2). They additionally outline the service's runtime to begin after a service is published or deployed. This agrees with Sommerville's (2006) and our classification, while contradicting Oracle's understanding of the order how to publish and integrate (or deploy) a service. Gu and Lago explicitly highlight design time and change time as necessary for a well-defined lifecycle approach. According to their approach, a service enters the change time after being provisioned and monitored. Service change takes place before reiterating the lifecycle, entering the requirements engineering phase. The approach does, except for the definition of change time and the total number of phases, match the classical SE lifecycle.

HP proposes a lifecycle with little information about the involved phases [Brauer and Kline 2005]. They define a triggering phase, in which the need for a certain service to be designed is determined and the design process is *trigger*ed. The following phases are *design*, covering the requirements analysis and design, and *source*, which covers the implementation and testing, followed by *deploy*, and *operate*. This lifecycle is the only one out of all presented lifecycles that implements a never-ending change time following the *operate* phase. Hence, a cycle reiteration is not possible. Compared to the classic SE lifecycle, the never-ending change phase maps to the reiteration of the complete cycle. The ability to retire a service or leave the cycle is not given.

Matsumura (2007) at WebMethods proposes a cyclic approach including change time and a change phase. Services are *plan*ned and *design*ed during design time. *Manage* covers deployment, while *run* and *use* correspond to the operation phase. For these phases, the authors lack to provide sufficient descriptions. After a service is run and used during runtime, it enters the change time. Herein, change covers, e.g., operational change and service version management. After the *change* phase, the cycle can be reiterated starting from the *plan* phase.

Authors at Systinet (2006) propose a cyclic lifecycle approach that implements the phases *design, build, deploy*, *assure,* and *change*. While the first three are similar to the SE lifecycle, *assure* corresponds to 'Operation/Maintenance'. Assuring a service can be understood similar to Oracle's *secure and manage* phase. If necessary, service corrections or adjustments are performed during the *change* phase within *change time*. Additionally, cycle reiteration is considered and starts with the *design* phase. The approach does not cover a requirements analysis phase.

The majority of the approaches considers phases that are comparable to the classic software engineering lifecycle (cf. Fig. 2). The differences originate from different understandings and differing levels of detail. Except for Oracle's lifecycle, design time and runtime cover comparable phases. Only two out of eleven approaches consider the possibility to leave the service lifecycle for service retirement. The main difference, however, is the inclusion of change time. Four out of eleven investigated approaches explicitly define change time, the remaining do not even mention it.

ANALYSIS AND DISCUSSION

Considering the definitions of the outlined approaches, we give short summaries for the super-phases design time, runtime, and change time in the following. Based on this, we discuss the relevance of each of these super-phases.

Design Time

Within the field of SE, design time covers the first four phases of the classical SE lifecycle: requirements analysis, software design, implementation/testing, and integration (cf. Fig. 1 and 2). The investigated approaches particularly emphasize the following aspects of design time. According to Gu and Lago (2007), it covers the service design satisfying the defined functional and non-functional requirements, refined service interfaces, and the style of interaction between services and their clients (asynchronous or synchronous invocation). Gu and Lago include deployment in runtime. Windley (2006) even defines a "deploy time". Most approaches, however, treat deployment as part of design time. Wall (2006a) emphasizes service categorization, modeling methodology, as well as service building and composing concepts as essential parts of design time. At Software AG (2008), design time comprises requirements analysis concerning the services' behavior, performance, quality-of-service (QoS), and security.

Summarizing, design time covers all activities of the software construction process that take place before and including service deployment.

Runtime

According to Sommerville (2006), runtime covers phase 5 of the classic software engineering lifecycle, *operation and maintenance*. This includes the use of the software, as well as maintenance and change management. Service monitoring is also part of this phase. Monitoring and reporting service-level agreement (SLA) behavior and the monitoring of other general performance aspects are also defined within the runtime phase [Matsumura 2007, Software AG 2008, and Wall 2006b].

To underline the customer's involvement during the usage phase, Windley (2006), Systinet (2006), Matsumura (2007), and Gu and Lago (2007) additionally present a consumer service lifecycle, which consists of the four phases *discover, bind, use,* and *disconnect* [Windley 2006]. *Discover* describes a service, capable of fulfilling the customer's demand, being discovered within the service provider's registry. *Bind* involves contract management by the provider and consumer, as well as the service level agreement handshake. The *use* phase describes the time until *disconnect* in which the service is actually used by the customer. If necessary, the customer lifecycle can be reiterated, starting at the *discover* state [Windley 2006]. Matsumura (2007) define the consumer service lifecycle to regulate transactions between service provider and service consumer. These interactions include broker transaction, data transformation, message queuing, and security handshakes. Authors at Systinet (2006) define the consumer service lifecycle involving the four phases *discover, bind, interact,* and *monitor*. *Discover* describes the provided service being discovered within the provider's registry. *Bind* involves contract management to negotiate the terms of use for a certain service, followed by the phase *interact*, where the service is used. Finally, in the phase *monitor,* details concerning the quality of service performance are collected and reported to the provider.

Concluding, the understanding of runtime basically covers service operation including monitoring. A consumer lifecycle is also part of this phase, covering activities and phases involving the service consumer starting in the service operation phase.

Change Time

Within the field of SE, change time is not explicitly considered. However, considerations for system improvements, error correction [Sommerville 2006], are mandatory.

Gu and Lago (2007) explicitly define a change time phase following the runtime phase. The trigger they mention is business requirements change which directly affects functionality adjustments. They argue that the ability to change services is required to meet user expectations and stay competitive. Nevertheless, they admit that a lifecycle reiteration is potentially necessary at this point in time. According to authors at Systinet (2006), change time covers service version management and service adaptation to newly encountered security requirements. Following change time, a service restarts processing the lifecycle. Matsumura (2007) emphasizes service version upgrades, operational change of services, service decommissioning, and service deprecation as central aspects of change time. In order to plan and design the outlined operational change, a subsequent reiteration of the cycle is required. According to authors at HP [Brauer and Kline 2005], change time is an endless adaptation process following the runtime phase. Change time involves service delivery, service delivery management, and service version management. A reiteration of the proposed lifecycle is not intended. Obviously, authors at Software AG (2007a) consider change time a crucial component of the service lifecycle. However, they fail to state its definition or purpose.

Given that only four out of eleven approaches consider change time, definitions vary considerably. Most of the authors, however, define change as additional phase following the classic operation phase. So according to these definitions, change time basically covers the management of service decommissioning, versioning, and retirement - including handling of functional changes, covering, i.e., service adoption, business process change, service portfolio change, technical requirements change, and security issues.

Discussion

Seven out of eleven approaches we reviewed include neither a change phase nor change time. Even the classic SE approach does not involve it, although change of software components or systems has always been an important topic in software engineering.

The original SE lifecycle covers software change and versioning, but does not define or require a change time phase in the corresponding lifecycle. Change is considered a sub-process of *Operation and Maintenance* and triggers a reiteration of the cycle. Every iteration of the lifecycle after the first one represents a software change process.

Basically, change time stands for and addresses the need to manage change activities that are required upon change requests originating from necessary functional changes. It implies addressing the change request in order to solve the occurred problem. As soon as the service itself is concerned, its requirements, definition, and design are reanalyzed – in order to finally accomplish code change with implies testing. Obviously, this is equivalent to reiteration of the service lifecycle, as defined by many of the approaches presented. In fact, Boehm (1988) described a spiral software development process that reiterates one cycle while addressing different evolving issues in each iteration. The approach by Brauer and Kline (2005) almost implements this by specifying a change loop.

Given that the operation and maintenance phase of the classic SE lifecycle includes change handling, the definition of a change phase or time seems unwarranted. It is assumed that it can safely be replaced by a cycle reiteration. Additionally to Matsumura (2007), Systinet (2006), and Gu and Lago (2007), this also exceeds the approaches by Woolf (2006), Strnadl (2007), Software AG (2008), and Brauer and Kline (2005), whose lifecycle does not reflect a cyclic behavior.

After sifting through and analyzing all these arguments, the authors of this paper cannot identify a clear motivation for a change time in a service lifecycle. We systematically disproved all arguments or advantages of such a super-phase. The imperative conclusion is the absence of the need to define and integrate a change time into a service lifecycle.

In the following, we define a service lifecycle that allows for the above considerations and extends the presented approaches while forming a generally applicable lifecycle.

A Service Lifecycle Model – Our Approach

Our service lifecycle approach mainly consists of seven phases (system analysis, service definition, service design, service development, service deployment, service operation, and service retirement). Excluding service operation, all phases are part

of design time – service operation is a runtime phase (cf. Figure 3). We define a feedback loop to allow for stepping back a phase for change purposes, e.g. due to an error that occurred in the preceding phase that must be corrected.

The entry phase is the *system analysis*, where fundamental decisions concerning the system and its scope are made. (Service granularity, e.g., is an aspect that increasingly gains importance when creating services. Many of the above investigated approaches awkwardly neglect an explicit *system analysis* or *requirements analysis* phase.) Once the *service definition* is accomplished, the *design* phase starts, where basic software design procedures concerning, e.g., interface definitions and design principles such as communication standards and code reuse are applied. The following phase is *service development* which basically covers the implementation and intensive service testing.

In service systems, we consider a distinctive *deployment* phase that requires much more attention than in component development or integration in SE. In addition to activities typically performed in the corresponding "integration" phase in the SE cycle, this phase covers final service tests, the installation on a server, which implies the delivery to the service host, registration in service registries or repositories, conformity checks to governance policies, and the resulting final approvals (e.g., description and interface checks).

Once a service is deployed, the service proceeds to the *operation* phase (runtime). This phase basically consists of service monitoring and change management. The latter covers functional change request handling, including service adoption, service versioning, business process change, service portfolio change, etc. In particular, it organizes the proper transition to the service *definition* phase when a change is requested. In this case, the service will be marked as deprecated and a definition process is triggered, i.e., it restarts traversing the lifecycle.

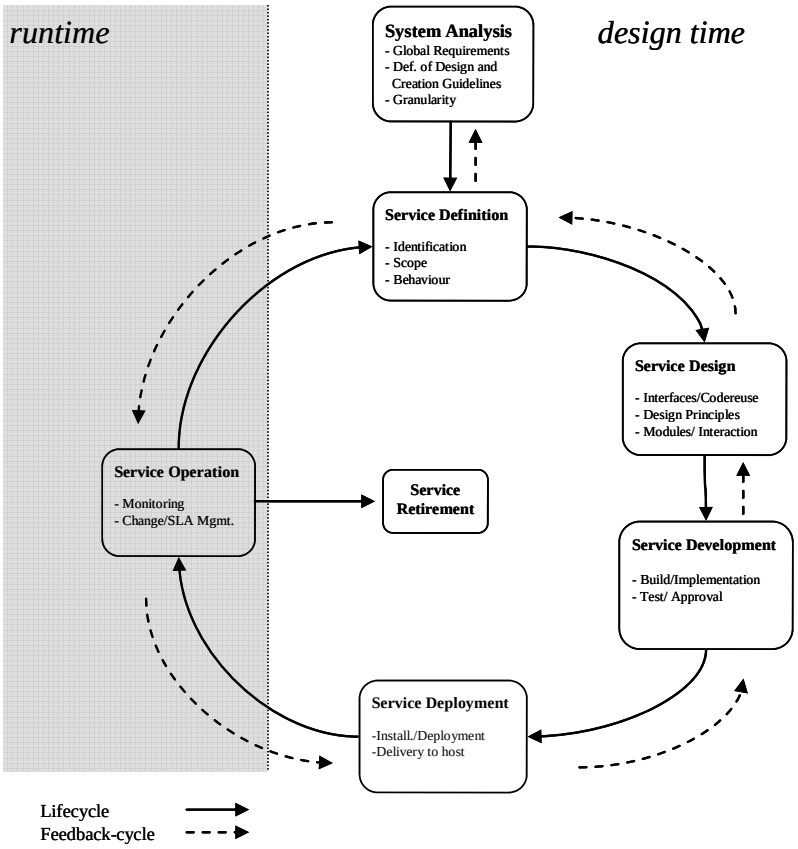

Figure 3. A Service Lifecycle – Our Approach

At this point, either a new service *version* is needed – or an additional *service*. In the latter case, a new service lifecycle iteration is started targeting the generation of a new service. In the first case, the service development process is forked. One service instance at its current version continues being operated and is marked as deprecated. A second instance is taken

offline and reiterates the lifecycle for improvement. When this instance has passed the service deployment phase, the old deprecated version is gradually replaced by the improved one, i.e., it traverses to the phase *service retirement*. This process of preparing and performing the reiteration of the lifecycle maps to the change time described in the approaches above – however, this process cannot explicitly be described as change phase or time.

This lifecycle model is congruent to the classic SE lifecycle and maps to all investigated approaches. It details most of the shorter cycles such as those by McBride (2007) and Windley (2006). As defined above, it covers the approaches by Woolf (2006), Software AG (2008), Brauer and Kline (2005), and Strnadl (2007), and extends them by the ability to reiterate the cycle. It also summarizes the longer cycles defined by Wall (2006a) and Gu and Lago (2007) by consolidating detailed sub-phases.

As demonstrated, our model maps to the lifecycles that define an explicit change phase or change time by addressing the necessity for change in the service operation phase and the reiteration of the lifecycle. In this respect, all phases in the change time box shown in Figure 2 become part of the phase Operation/Maintenance – the remainder of the change activities are covered by reiterating the service lifecycle.

CONCLUSION

In this paper, we investigated eleven different approaches to the service lifecycle.

A central aspect of service governance is the management and definition of a consistent service lifecycle. As shown in this paper, approaches can vary from one extreme to another in structure and definitions. However, almost all approaches contain particularities that reflect experience in addressing the specific needs of a SOA system. Our life cycle approach combines them with the common SE perspective and a discussion that invalidated the need for an explicit change phase or change time. However, each of the investigated lifecycle proposals maps to our model. We elaborated a consolidated approach that integrates all hitherto existing findings and allows for the particular requirements of a SOA system.

In spite of the unusual phase names (assure meaning operation, and manage being either design or runtime activity) in the compared lifecycle models, we noticed that all change phases and change time definitions can completely be replaced by the common definitions of the phase *Operation/Maintenance* combined with lifecycle reiterations. There simply seems to be no need for such a time as “change time”.

DISCLAIMER

This work is supported in part by the E-Finance Lab e. V., Frankfurt am Main, Germany (www.efinancelab.com).

The project was funded by means of the German Federal Ministry of Economy and Technology under the promotional reference 01MQ07012. The authors take the responsibility for the contents.

REFERENCES

1. Afshar, M. (2007) SOA Governance: Framework and Best Practices. Oracle Corporation Headquarters, http://www.oracle.com/technologies/soa/docs/oracle-soa-governance-best-practices.pdf, visited January 15, 2009.
2. Bhushan, B. and Tayal, S. (2007) Software Engineering, Laxmi Publications, ISBN: 978-8131804797.
3. Boehm, B.W. (1988). A spiral model of software development and enhancement. IEEE Computer, 21(5), 61-72.
4. Boehm, Barry W. (2007) Software Engineering: Lifetime Contributions to Software Development, Management, and Research, Wiley & Sons, ISBN: 978-0470148730.
5. Brauer, B. and Kline, S. (2005) SOA Governance: A Key Ingredient of the Adaptive Enterprise, Hewlett Packard 2005, https://h30406.www3.hp.com/campaigns/2005/events/infoworld_soa_forum/soa.php, visited February 16, 2009.
6. Clark, Andrew J. (2005) IT Governance: Determining who decides. EDUCAUSE, Center for Applied Research. Volume 2005, Issue 24, http://net.educause.edu/ir/library/pdf/ERB0524.pdf, visited February 20, 2009.
7. Gu, Q. and Lago, P. (2007) A stakeholder-driven service life cycle model for SOA, in *IW-SOSWE '07: 2nd international workshop on Service oriented software engineering*, New York, USA, pages 1-7.

8. Huhns, M. and Singh, M. (2005) Service-oriented computing: key concepts and principles, IEEE Internet Computing, volume 9 (1), pages 75–81, Jan-Feb 2005, http://www.cse.sc.edu/~huhns/journalpapers/V9N1soc.pdf, visited February 16, 2009.
9. Jacobson, I. and Bylund, S. (2000) The road to the unified software development process, Cambridge University Press.
10. Kjaer, A. M. (2004) Governance (Key Concepts), Polity Press, February 2004, ISBN: 0745629792.
11. Krafzig, D., Banke, K., and Slama, D. (2004) Enterprise SOA: Service-Oriented Architecture Best Practices (The Coad Series), Prentice Hall PTR, 2004.
12. McBride, G. (2007) The role of SOA quality management in SOA service lifecycle management, IBM DeveloperWorks, 2007, http://www.ibm.com/developerworks/rational/library/mar07/mcbride/, visited December 20, 2008.
13. Matsumura, M. (2007) The definitive guide to SOA Governance and lifecycle management, WebMethods 2007, http://www.scribd.com/doc/7056281/Guide-to-SOA-Governance, visited February, 8 2009.
14. Mukhi, N. K. and Plebani, P. (2004) Supporting policy-driven behaviors in web services: experiences and issues, in *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 322–328, New York, NY, USA, 2004, ACM.
15. Newcomer E. and Lomow G. (2004) Understanding SOA with Web Services (*Independent Technology Guides)*, Addison-Wesley Professional, December 2004.
16. Niemann, M., Eckert, J., Repp, N. and Steinmetz, R. (2008) Towards a Generic Governance Model for Service-oriented Architectures. In: Association for Information Systems (AIS): Proceedings of the Fourteenth Americas Conference on Information Systems (AMCIS 2008), Toronto, ON, Canada, 2008, AIS, August 2008.
17. Papazoglou, M. P. (2003) Service-Oriented Computing: Concepts, Characteristics and Directions, in *4th International Conference on Web Information Systems Engineering (WISE 2003)*, Rome, Italy 2003, pp. 3-12.
18. Software AG (2007a) CentraSite Governance Edition (2007), http://www.softwareag.com/corporate/images/sag_centrasite_gov_ed_fs_dec07-web_tcm16-33856.pdf, visited February 16, 2009.
19. Software AG (2007b) "Rule your SOA", Software AG, http://www.softwareag.com/Corporate/Images/WP%20SOA%20Governance_tcm16-22130.pdf, visited February 2, 2009.
20. Software AG (2008): SOA Lifecycle Governance, Software AG, http://documentation.softwareag.com/webmethods/soal8/overview/soa.htm, visited February 19, 2009.
21. Sommerville, I. (2006) Software Engineering, Addison-Wesley Longman, Amsterdam.
22. Strnadl, C. F. (2007) Bridging architectural boundaries design and implementation of a semantic BPM and SOA Governance Tool, in *ICSOC '07: Proceedings of the 5th international conference on Service-Oriented Computing*, pages 518–529, Berlin, Heidelberg, 2007, Springer.
23. Systinet (2006) SOA Governance: Balancing Flexibility and Control within an SOA, http://www.vivit-worldwide.org/uploadedFiles/Product_Centers/Service_Oriented_Architecture_(SOA)/09EX001HPMAY07_Mercury-Systinet_SOA_Governance.pdf, visited February 16, 2009.
24. Wall, Q. (2006a) Understanding the Lifecycle within a SOA: Design Time. Oracle Corporation, http://www.oracle.com/technology/pub/articles/dev2arch/2006/08/soa-service-lifecycle-design.html, visited February 19, 2009.
25. Wall, Q. (2006b) Understanding the Lifecycle within a SOA: Runtime, Oracle Corporation, http://www.oracle.com/technology/pub/articles/dev2arch/2006/11/soa-service-lifecycle-run.html, visited February 19, 2009.
26. Weill, P. and Ross, J. (2004) IT Governance: How Top Performers Manage IT Decision Rights for Superior Results, Harvard Business School Press, June 2004.
27. Windley, P. J. (2006) SOA Governance: Rules of the Game, in: *InfoWorld, v28 n4 p29(5)*, January 2006, http://akamai.infoworld.com/pdf/special_report/2006/04SRsoagov.pdf, visited February 8, 2009.
28. Woolf, B. (2006) Introduction to SOA Governance, International Business Machines Corporation (IBM) 2006, http://www.ibm.com/developerworks/library/ar-servgov/, visited February, 16 2009.