Software Engineering Services

Ingo Schnabel
itestra GmbH
Ludwigstr. 35
Germany – 86916 Kaufering
Phone/Fax: +49 (8191) (97075) {85/88}
schnabel@itestra.com

Markus Pizka
itestra GmbH
Ludwigstr. 35
Germany – 86916 Kaufering
Phone/Fax: +49 (8191) (97075) {85/88}
pizka@itestra.com

Abstract *Mapping business processes on information technology through service oriented architectures is an increasingly important topic both for business and technology people. While SOA is not very novel from a technical point of view (see J2EE, RPC, CORBA), the service paradigm may significantly influence the orchestration of business processes and increase the productivity, agility and flexibility of enterprises in diverse domains. Outsourcing in the automotive industry as well as globally operating call-centers show how to successfully implement services with defined service quality. So far, the software community hardly discusses how to apply the service paradigm for software development itself. For example: What kind of services exist in typical software development projects and processes? How are service contracts specified and negotiated? How can services be governed? Due to the fact that there are enormous differences in productivity, agility and flexibility between different processes, organizations and developers, the software industry would strongly benefit from a proper service orientation. This paper explains how software engineering services can be used to make the productivity (i. e. time and cost) in software projects as well as the quality of the outcome more predictable and comparable.*

Keywords: software engineering, service orientation

1 Service Orientation Matters

Over the last 30 years, software evolved from helpful little programs with some thousand lines of code to complex information systems with million lines of code. These information systems have a growing impact on competing businesses. Developing and maintaining large-scale software systems is not comparable with writing a thousand lines of code program. 30 years ago programs were written to improve business processes and the economic loss created by a failing software project was rather small. Nowadays business processes strongly depend on software; e. g. stock markets are inconceivable without large-scale software systems. Because large-scale software engineering requires great expense for business, a failing project is a serious threat for stakeholders. Therefore stakeholders have a major interest in controlling the costs of software engineering and recognize eventual risks as early as possible. How are costs controlled in real-life projects today? Most times, the performance of a project (and also software vendors) is measured by accumulating daily-rates of staff and comparing this with estimated. If estimated man-days correspond to consumed man-days the project is regarded on schedule. As a consequence the earliest point of time to control the outcome of a project is -even in iterative and incremental processes- the acceptance test. Reports like the CHAOS report [1] substantiate that there is still an outstanding number of large-scale software projects which fail

to deliver in time and budget. It is obviously not possible to recognize time and cost overruns at an early phase of a project by measuring performance in man-days. In addition to this, measuring cannot indicate the reason for the occurring problems; i. e. no reasonable countermeasures for project management can be proposed. Furthermore measuring performance in man-days assumes that every team member achieves an equal level of performance. Unfortunately this assumption is absolutely wrong, e. g. best programmers are up to a hundred times more productive than the worst [2, 3, 4]. In order to solve these problems we think that the outcome of each development activity has to be verified against a priori determined properties. How can such desirable properties be defined and ensured? Outsourcing in other lines of business has similar difficulties: how can the performance or the quality of the outsourced service be assured? The Automotive industry, globally operating call-centers and others have shown how to successfully implement outsourcing by applying the service paradigm. It stands to reason to transfer this proved concept frequently promoted from software people to businesses to software engineering itself.

2 Related Work

The work described in this paper has obviously a strong relationship with service orientation, which is discussed from a technical and a business point of view [5, 6, 7].

3 Services in Software Development

Software development is - according to the definition of services - a service. A service is characterized by a set of key attributes:

- Intangibility
- Perish-ability
- Lack of transportability
- Lack of homogeneity
- Labor intensity
- Demand fluctuations
- Buyer involvement and client-based relationships

Because not all kinds of services can fulfill the key attributes, business theorists refer to a continuum between pure commodity goods and pure services as a service good continuum, illustrated in figure 1. As software development is by far more a service than a commodity good, defining software engineering services seems to be suitable. The service paradigm

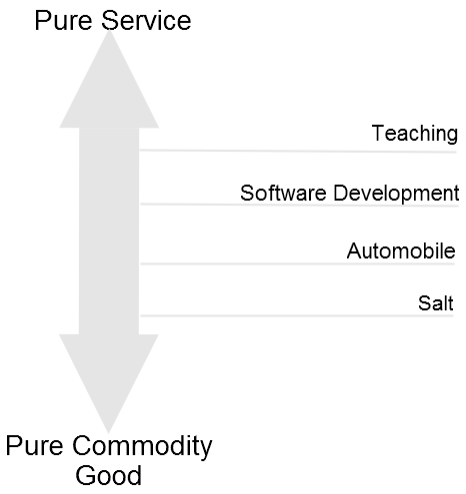

Figure 1: Service-Good Continuum

aims to make the performance of activities comprehensible by defining service contracts and service level agreements; i. e. a customer may finally buy well-defined services from one or more vendors. Defining proper service contracts for a unified “software development service” is impossible, because of its complexity. The idea proposed in this paper is to separate the one complex “software development service” according to development activities into smaller services with well defined service contracts.

3.1 Separation of Concerns

The service paradigm itself is closely related to the well-known separation of concerns (SoC) principle [8, 6]. The purpose of SoC is to break a certain concept into small entities with minimal overlapping responsibilities. The strict separation of responsibilities enables services to define revisable properties of the service outcome. A service specification is a description of what the service should perform. An important part of the service specification is the service contract, which is a formal agreement that determines properties of the service in- and output. Service contracts are negotiated between service consumers (stakeholders) and service providers (software vendors). In this sense, successfully applied software engineering services structure a complex development process into well-defined services.

3.2 Revision of the Service Outcome

As the definition of service contracts enables the revision of software development services, single software development activities become comprehensible and assessable. One part of the service contract is the service level agreement, which determines non-functional requirements of the service; e. g. three errors are acceptable in thousand lines of code, the service has to be completed within five days et cetera. After a service is completed, the outcome is verified against the service contract. It is accepted if the outcome meets the service contract, otherwise it is rejected. Furthermore the revision entails two other vantages:

1. The fact, that software engineering services correspond to development activities like business analysis, programming or testing, allows project managers to identify activities which fail to achieve the required performance early.

2. The assessment of team members in real projects is often more based on subjective judgment than on clear facts. Introducing service orientation into a project helps to make the productivity of single team members traceable, because every activity is controlled by the revision of the service.

4 Establishing Software Engineering Services

The model of a software engineering service is depicted in figure 2. The input of a service is a set of artifacts, e. g. requirement documents. The service processes the input according to its contract. During the processing of the service new artifacts are created or input artifacts are changed and committed to the output. The processing of services requires a certain amount of resources. For example, a refactoring service needs two programmers, two licenses of a specific refactoring tool, two workstations and so on. How many and what kinds of resources are used by a service depends on the definition of the service contract. The specification of services is determined by the customer's demands, business processes and goals of the project. For example, the quality assurance services needed for the navigation software of the mars exploration rover differ from those quality assurance services needed for an Internet weather forecast service. In the following sections we explain

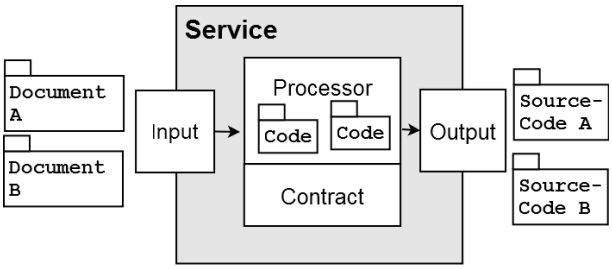

Figure 2: Service Model

how software engineering services are specified and established in software projects. Thereby we concentrate on the definition of service contracts - and on service level agreements as an integral part of the service contract. Afterward we explain, how a service verifies input and output artifacts against the service contract,

in order to enable the service provider to ensure the performance of the service. But first of all it has to be explained, how the complex overall "software development service" can be divided into manageable services.

4.1 Categorization of Software Engineering Services

Software development activities are in general related to three domains: development, management-centric and maintenance activities. The categorization of software engineering services, illustrated in figure 3, reflects these domains. The services applied by a project, strongly depends on the project itself and on the needs of the stakeholders.

- **Development Services** Services concentrating on the design and implementation of software. This includes requirements engineering as well as developing test cases.
- **Management-Centric Services** Services which are strongly linked with process management activities; project management, change control management, test management or quality assurance.
- **Operation and Maintenance Services** These services are applied during or after the software is released and deal with the restructuring and the analysis of the software system.

4.2 Specification of Services

As the benefit of software engineering services depends on the adequacy of the service contract for the service performance, defining a service contract is crucial for the success of a service and must not be underestimated. Although service contracts are not defined from scratch for each project, it has to be taken into account, that different kinds of projects require different service contracts. Jones describes the definition of proper enterprise services in [5] in four steps. We adapted this process for software engineering services:

1. **WHAT** Determine, what the service and its in- and output is.
2. **WHY** Identify, why a service is required.
3. **WHICH** Which resources are required to perform the service?
4. **HOW** How can a service contract be defined?

First, it is important to figure out, what the service actually should perform and determine the service in- and output. Since the performance of a service depends on the output, the output - usually in form of artifacts - is an eminent part of the contract. Second, because of costs (time and budget) it is important to justify, why a specific service is required. Third, before a service can be established, it has to be examined whether all resources needed to perform the service successfully are available. Fourth, the service contract has to be negotiated satisfying both the service consumer and the service provider.

4.3 Service Contract

A service contract is a negotiated agreement between the service consumer (the stakeholders) and the service provider (software vendors). As depicted in figure 4, a ser-

Figure 4: Service Contract

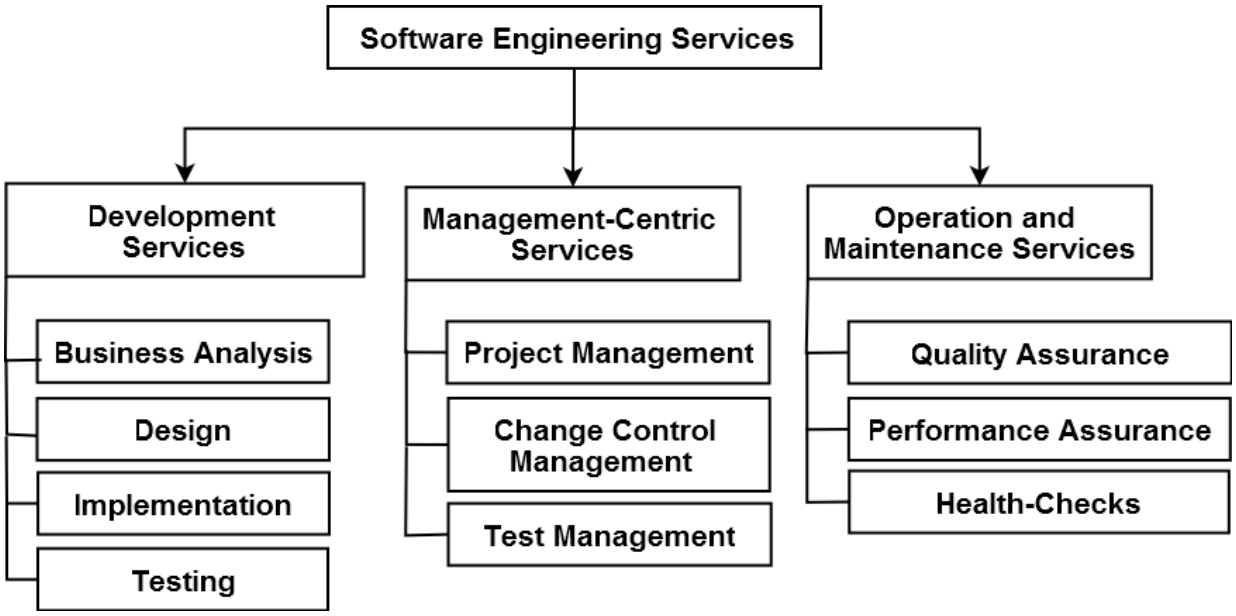

Figure 3: Software Engineering Services

vice contract consists of a header, functional-requirements, a service level agreement, and a specification of the associated artifacts.

The header consists of the service name (e. g. Business Analysis, Design and Implementation, Documentation, Project Management) and an informal description, that indicates in general terms what the service is capable of. Functional requirements determine the functionality of what the service exactly accomplishes by means of a declarative description. Although functional requirements vary significantly from service to service, quality properties of artifacts can be described in some terms:

- **Redundancy Free and Consistent** Redundancy is a major problem in artifacts that change over time, like code, requirement specification or documentations. A change in one place of the artifact entails changes in others places. A violation of consistency causes in any case enormous costs and makes artifacts error-prone.
- **Completeness and Level of Abstraction** Practical experiences show that, choosing the right level of abstraction for an artifact is not easy. Often, some aspects of the artifact are described in a very detailed way, other parts are described high level or are not considered at all. The consequence is that artifacts appear to be premature and incomplete.
- **Correctness** The objective of any activity is the correctness of the created artifacts.
- **Structure** Well structured artifacts not only ease understanding of artifacts, they also help to fulfill the preceding attributes.

The artifact specification determines the document format and the properties of artifacts; e. g. the header of each source file contains the name of the author, the creation date and an abstract of the source file. Resource specification is a compilation of all required resources to process a service; e. g. human, hardware and software resources. Service level agreements (SLA) are requirements not directly depending on the specific activities of the service. A SLA defines the quality properties of the service by means of measurable criteria , i. e.the SLA determines, if the outcome of a service can be accepted or has to be rejected. SLAs are defined with respect to the SMART-principle:

- **Specific** Every service needs unambiguous goals.
- **Measurable** Every criterion of a service level agreement must be measurable.
- **Achievable** Every goal of a service must be achievable with respect to the required resources.
- **Relevant** Irrelevant goals increase the costs of services without increasing their benefit.
- **Timely** Every service must be finalized within a certain time frame.

4.4 Revision of Services

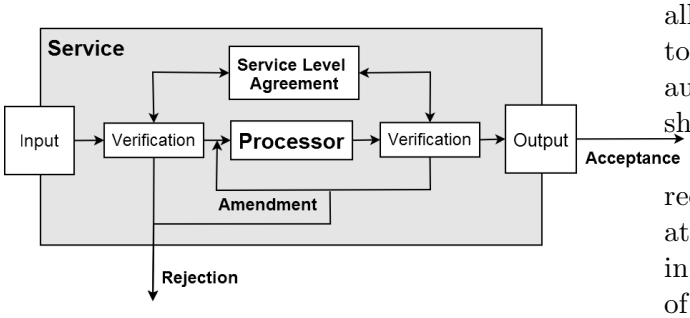

Figure 5: Service Verification

As mentioned above, the service level agreement plays a major role in making the outcome of services revisable, because it provides a basis for revisable artifacts. The functional requirements and the service level agreement determine the verification of the service output. Figure 5 illustrates how a service is verified against the SLA. First, the service checks, if input artifacts violate the service level agreement. In case of a violation the service is rejected. Otherwise the service processes the input artifacts and generates the output artifacts (explained in figure 3). Then the output artifacts are checked for a SLA violation. If the SLA is violated (e. g. a time overrun or a violation of functional requirements) according to the SLA the service is rejected or the outcome has to be mended. Otherwise the service is accepted.

4.5 Example of a Testing Service

In the following, we explain how a service can be specified by means of an example. The initial situation is, a software vendor asked for a software testing. The customer wants to perform a system test on a software system with about two hundred thousands line of code. The specification of the service starts with the four step process described in section 4.2:

What is the service and its in- and output? The goal of the software testing service is to validate the exterior quality (see [9]) of the software; i. e testing correctness, reliability, usability and integrity. The service input is a compilable and runnable software system. The service output is a compendium of all located errors. The software should be tested manually through test cases determined by the customer. Furthermore it is required to deliver automatic unit tests. The automatic unit test should cover about 60% of the code.

Why is the service required? The service is required to pass the acceptance test and to create automatic unit tests. These tests are used in the further development and in system tests of later releases. The acceptance test distinguishes between major and minor errors. The acceptance test fails as soon as any major error occurs.

Which resources are required to perform the service? In order to perform the service test engineers, the complete compilable source code, computers, and a development environment (for the unit test) is required.

How can the service contract be defined?

- **Header**
  System Test Service: The service tests manually a given software through test cases a priori defined by the service consumer.
- **Functional Requirements**
  1. Locate and correct all errors by executing the test cases.
  2. Write an automatic unit test that covers 60% of the source code (in the

modules *Algorithm* and *Persistence* the unit test must cover more than 99%).

- **Artifact Specification**
  The service output is a document which lists all located and corrected errors. There are no formal requirements on the document format.

- **Resource Specification**
  The service demands five test engineers. Every test engineer must be able to execute all test cases and develop unit tests on his own workstation.

- **Service Level Agreement**
  The service is rejected if one of the following conditions is violated:

  1. No major error occurs in the software.
  2. One minor error occurs in 1000 lines of source code.
  3. Unit tests cover at least 60% of the source code. The modules *Algorithm* and *Persistence* must have a code coverage of more than 99% .
  4. All resources determined in the resource specification are available.
  5. The service creates all output artifacts determined in the artifact specification.
  6. The system test expense is not more than half of the implementation expense.

5 Practical Experiences

Software development services as sketched in this paper have already proved their benefits in commercial large-scale software projects[1].

[1]Due to non-disclosure agreements, no details that would allow to draw conclusion about the site and purpose these projects can be published.

One example is a large-scale project with a total of more than 60 man-years and critical interest to the stakeholders. As the software was much more used than originally assumed, after one year stakeholders wanted to improve the overall performance of the software. It soon became clear, that a performance improvement could be achieved by removing some bottlenecks, but required a complete redesign of the source code. Because stakeholders had objections to a complete restructuring of the source code, we suggested to establish two development services, a performance analysis and a performance improvement service. First we used the performance analysis service to identify the parts of the source code that had to be redesigned. As negotiated in the SLA, stakeholders received a document explaining all necessary activities for the performance improvement. Then the performance improvement service realized the redesign of the source code. Afterward the performance analysis service was used to substantiate the performance improvements.

The stakeholders are highly satisfied with the results of the software engineering service, because they understand both all necessary activities of the service and all arising expenses. As part of this success, the stakeholders decided to proceed with the software engineering services.

6 Conclusion and Future Work

This paper introduced the service paradigm to software development: we call such services software engineering services. Clearly, software engineering services do not substitute conventional software processes. In fact, software development services support processes in making the outcome of the process measurable.

The key contribution of software engineering services is the definition of a service contract. The service contract specifies functional requirements, in- and output artifacts, required resources and a service level agreement. The

service level agreement determines with respect to the service if the service outcome is accepted or rejected. Services thereby enable process control to measure the performance of development activities; i. e. project control is no longer based on estimated progress and man-days, but gets a precise instrument for measuring performance.

Our future work will concentrate on the development of a software engineering service infrastructure. The infrastructure aims to enable service government and to ease the interaction between different software engineering services.

References

[1] The Standish Group International Inc. Chaos, 1995.

[2] B. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.

[3] Jack Greenfield. The case for software factories. *Microsoft Architects Journal*, (3), July 2004.

[4] Robert C. Martin. One per pixel, 2003.

[5] Jones Steve. *Enterprise SOA Adoption Strategies*. Lulu.com, 2006.

[6] L. Cherbakov, G. Galambos, R. Harishankar, S. Kalyana, and G. Rackham. Impact of service orientation at the business level. *IBM Syst. J.*, 44(4):653–668, 2005.

[7] J. Allen, I. Barnes, S. Bennet, D. Groves, F. Lebegue, R. Ramanathan, and D. Slaughenhaupt. Realizing the business benefit of service-oriented architecture, 2005.

[8] Edsger W. Dijkstra. On the role of scientific thought. In *Selected Writings on Computing: A Personal Perspective*, pages 60–66. Springer-Verlag, 1982.

[9] William C. Hetzel and Bill Hetzel. *The Complete Guide to Software Testing*. John Wiley & Sons, Inc., New York, NY, USA, 1991.

[10] Markus Pizka and Andreas Bauer. A brief top-down and bottom-up philosophy on software evolution. In *Proc. of the Int. Workshop on Principles of Software Evolution (IWPSE)*, Kyoto, Japan, September 2004. IEEE Computer Society.