Citace:

BRABEC, Tomáš, BUCHALCEVOVÁ, Alena. Suitability of XP for service-based application development. Praha 10.06.2007 – 12.06.2007. In: Systems Integration 2007. Praha : KIT VŠE, 2007, s. 109–116. ISBN 978-80-245-1196-2.

Suitability of XP for service-based application development

Tomáš Brabec, Alena Buchalcevová

Dept. of IT, Prague University of Economics
Winston Churchill sq. 4
130 67 Prague 3
brabec@vse.cz, buchalc@vse.cz

Abstract
In recent decade the process of developing of new software product versions has speed up rapidly. The necessity of flexible and particularly prompt responses to the changes triggered off genesis of new technologies, software architectures and methodologies. One of the most significant new concepts in IS/ICT became services. The Service Oriented Architecture (SOA) allows defining services operation environment, web services then form one of the available technologies for SOA realization. The rigidity of original plan-driven methodologies for software development limits the possibilities of adapting the development process to the changes and up-to-date requests. Nevertheless, this problem might be solved either by adoption of any agile methodologies – represented among others by eXtreme Programming (XP) – or by updating original plan-driven methodology with agile principles – as e.g. the RUP does. This article contributes on the theme how XP supports service-based application development comparing to SOA plug-in for RUP.

Keywords
SOA, eXtreme Programming, RUP, Web services, Service-based application life cycle.

1 Introduction

During past decade significant changes have occurred within the approach to the business application development. There are many causes, to the most considerable belong *continual changes* in the application environment (business areas and goals, legislation, competition, custom practices, preferences and demands of customers) and necessity to respond immediately, effort to *utilize* off-the-shelf systems, applications and components as well as the need for rapid and easy *integration* of new customers and suppliers into both business processes and supporting information systems.

The integration demand of external and internal subjects into corporate business has resulted in rejection of distributed applications with tight coupling, such as CORBA or DCOM, and opened territory for new open standards and technologies.

The effort to high utilize present software products in conjunction with integration demands caused the enforcement of service oriented architecture (SOA) that uses loose couple to bind individual elements. Many applications and information systems have started to operate in the form of services or at least to employ services. Thus services became fundamental building blocks for these

applications and systems. Web services then represent advisable technology to implement application based on SOA.

As a result of continuous changes and the need to respond to them new kind of methodologies – so called agile – appeared. Existing plan-driven methodologies based upon strict, exactly followed processes and policies, turned up inflexible. Agile methodologies attempt to minimize risk by developing software in short timeboxes, called iterations, emphasize real-time communication and working software as the primary measure of progress. With respect to that many originally plan-driven methodologies started to adopt agile principles, in the first place short iteration cycles and periodic updates of project plans.

1.1 Service Oriented Architecture

Service Oriented Architecture (SOA) defines usage of loosely coupled software services to support the requirements of business processes and software users. In this architecture two computing entities, such as programs, interact in such a way as to enable one entity to perform a unit of work on behalf of second entity. Similarly the second entity could provide certain functionality to third entity and become all at once provider even consumer of a service.

Service interactions on lowest – message – level are defined using a description language (mainly WSDL) that describes the public interface, protocol bindings and message formats required to interact with a web service. Conversation interaction of services can be captured with process languages like BPEL. Each interaction is self-contained and loosely coupled, so that each interaction is independent of any other interaction. SOA also attends to the way that services are described and organized to support real-time automated discovery in repositories (e.g. in UDDI registry) and usage of suitable services.

There are several conditions that must be fulfilled to successfully operate SOA [5]:

1) All functions (business functions, business transactions composed from low-level functions, system functions) are defined in the form of services.
2) All services are independent and outward act as black-boxes. External components do not care how they are implemented internally if they return expected results.
3) In the most general sense, the services are invokable. At an architectural level, it is irrelevant, whether services are local or remote or what interconnect scheme and protocols were used.

1.1.1 SOA Life Cycle

High et al. in [4] start SOA life cycle with modeling the business (Model phase) and continue with translating the model into an information system design (Assembly phase), deploying the system (Deploy phase), managing that deployment and using the results coming out of that environment to identify ways to refine the business design (Manage phase). These four phases are cycled into iterative steps. The life cycle is then layered on a backdrop of a set of governance processes (Governance & Processes).

Model

Primary purpose of this phase is to capture business design, translate that into a specification of business processes and their activities, goals and assumptions and thus create a model of business. Activities will be within SOA architecture realized as services. The model should also capture business metrics to measure performance and efficiency of the business.

Outside of documenting current business architecture the model can be used to simulate how business processes will actually run and to optimize them. During this phase we should obtain the answer to question of what kind of services will we need and what data will services work with.

Assembly

The goal of this phase is to assemble the information system artifacts that will implement business schema resulted from previous phase. The business design is converted into a set of business process definitions and activities deriving the required services from the activity definitions.

Existing asset inventories (legacy programs) are searched to find application components that already meet the needs. In some case an adoption for new environment is needed.

Deploy

During this phase the hosting environment (conforming integration and security requirements) for applications is created and those applications are deployed. This includes the application's resource dependencies, operational conditions, capacity requirements, and integrity and access constraints. The techniques for ensuring availability, reliability, integrity, efficiency and service ability are to be considered.

Manage

This phase is focused on maintaining the operational environment and the policies expressed in the assembly of the SOA applications deployed to that environment. This includes monitoring performance of service requests and timeliness of service responses; detect failures in various system components; detecting and localizing those failures; routing work around them; recovering work affected by those failures; correcting problems; and restoring the operational state of the system.

Managing includes also tuning the operational environment to meet the business objectives expressed in the business design, and measuring success or failure to meet those objectives.

Governance & Processes

The processes of this phase ensure that compliance and operational polices are enforced, and that change occurs in a controlled fashion and with appropriate authority as envisioned by the business design.

2 Service-based application life-cycle

Service-based application life cycle phases are similar to other types of applications. We can find Analysis & Design phase, Realization phase, Deployment phase and Operation & Maintenance phase. However, the content of those phases is rather specific, related just to the services nature.[1] (The activities and deliverables of individual phases are described in Table 1.)

Analogous to the life cycle of whole SOA (see [4]) either a service-based application life cycle must involve a set of processes focused on Control & Governance. Such a phase guarantees that changes to an application will be introduced in controlled manner. It is about establishing who has the authority, and the processes they use, to decide what changes will be made. Processes and subprocesses for decision making must be defined (including the escalation paths for resolving conflicting decisions and goals) as well as a blend of policy against which change must conform ([4]).

2.1 Application types

As a matter of form we can identify three types of applications to be built on web services:

[1] One can point out the requirements analysis phase is missing. With respect to the SOA, we may assume that business design and business processes models contain preliminary requirements and jobs for the analytic phase.

I. Composite application that is a set of related and integrated services that support a business process built on SOA ([4]).

II. Single service application formed by an elementary service that represents an access point to a specific functionality and that does not rely on any other service.

III. Adapter like application representing a service used to make accessible legacy systems by adapting their incompatible interfaces in such a way that application clients are able to avail.

We have to say, of course, that composite applications may be composed from services of another two types to form a service with higher value.

2.2 Life-cycle phases

Due to the different nature of service-based application types mentioned in 2.1 individual life cycle phases differ both in the activities to be undertaken and their contents. A brief summary brings Table 1.

Phase	Deliverables and activities	Application type		
		I	**II**	**III**
Control & Governance	Organizational aspects – planning and deliverables schedule; project team and roles establishing; definition of both decision processes and guidelines to respond to special and specific situations; setting up a policy against which all future changes must conform.	x	x	x
	Control aspects – governance, management and regulation of life-cycle.	x	x	x
Analysis & Design	Analysis of either requirements or process operations to be realized by services.	x	x	
	Analysis of legacy system interfaces and functionality.			x
	Identification and hierarchical categorization of services ([1]).	x		
	Partitioning service portfolio[2] ([6]).	o	o	o
	Specification of integration needs and patterns ([3]).	x	o	o
	Detailed specification of services (structural specifications, behavioral spec., policy spec.) and individual components.	x	x	x
	Design of interfaces, messages structure and format.	x	x	x
	Model of service interactions within realized process (BPEL), service choreography and coordination (WS-Choreography), transactions (WS-Transaction).	x		
	Security model (WS-Security).	x	x	x
Realization	Implementation of new services including description and semantics.	o	o	x
	Localization of proper existing services, either internal or external; contract with service provider (QoS, SLA).	o	o	
	Refactoring existing services from their original form to fit the	o	o	

[2] Organizing (recently) identified services into logical partitions (without needing the services to be „owned“ by any one partition) to be available for other projects.

Phase	Deliverables and activities	Application type		
	desired form ([6]).			
	Assignment of services and components into appropriate places within SOA.	o	o	o
	Services and components functional testing.	x	x	x
	Assembly.	x		
	Integration testing.	x	o	o
Deployment	Deployment into a secured and integrated environment, relevant data migration.	x	x	x
	Availability and performance testing.	x	x	x
	Service publishing.	o	o	o
Operation & Maintenance	Monitoring performance of service requests and timeliness of service responses.	x	x	x
	Maintaining problem logs to detect failures.	x	x	x
	Detecting and localizing failures; routing work around them; recovering work affected by failures.	x	x	x
	Tuning the operational environment to meet the business objectives.	x	x	x
	Routine maintenance, administering and securing of applications and users.	x	x	x
	Feedback evaluation and continuous business process improvement.	x		

Table 1 - life cycle activities and deliverables

Due to the limited scope of this contribution the Table 1 lists only key activities and deliverables with no detailed description. Character „x“ in the column „Application type“ stands for that performing an activity is essential, „o“ stands for an optional (might be beneficial).

3 The level of support of application life-cycle by XP methodology

SOA and web services originated from the need to accept a call of unsteady and versatile business environment with the aim to allow enterprises faster and accordingly respond to the business changes while achieving maximum reusability of existing software. Likewise agile methodologies aim to quickly and flexible respond to the requirement changes.

The eXtreme Programming methodology (see [1], [5] for more details), probably the best known agile methodology in CR, builds on simplicity (the smallest but still functional deliverable), small versions, metaphor (XP equivalent of an analysis), design[3] and its everyday improvement, continuous testing, periodic source code revisions, pair programming, collective ownership of code, continuous integration, customer attendance in the workplace, 40-hour working week and standards for writing code.

Basic activities of XP are: Planning, Designing, Coding, and Testing. With regard to described properties of XP and to the fact that XP represent “Test-Driven-Development” approach, one can with

[3] XP does not have isolated steps of analysis and design, both are solved within an implementation step.

some exaggeration claim that any iteration of design and implementation is done only to such a degree we are able to run an appropriate test.

In our evaluation of the level of XP support for development and operation of service-based applications we start from the service-based application life cycle described in section 2.2, or with life-cycle activities, to be more precise. For every activity/deliverable we consider how pure XP (with no additional modifications) solves or supports that activity in comparison the RUP methodology enhanced with special SOA modeling plug-in.

Activity/deliverable	**RUP for SOA[4]**	**XP[5]**
Organizational aspects	The *Project Management* and *Configuration & Change Management* disciplines involve activities, artifacts and workflow to fulfill "Control & Governance" life-cycle phases aspects.	Version plans[6]; team composition; standards for writing code.
Control aspects		Collective ownership; Planning game control phase; customer in the workplace; collective appraisal and decision making of changes.
Requirements or process analysis	The *Requirements* and *Analysis & Design* disciplines cover tasks of business processes and legacy systems analysis.	Research and Engagement steps of Planning game; initial architecture design and its continuous improvement.
Legacy systems analysis		-do-.
Services identification and categorization	Supported by the *Identify Services* activity of RUP's SOA plug-in belonging to the *Analysis & Design* discipline.	Partial support within design step; no appropriate method is available.
Partitioning service portfolio		No explicit support.
Specification of integration needs and patterns	Supported by those activities of the *Implementation* discipline related to the integration.	No explicit support.
Detailed specification of services	Supported by the *Service Design* activity of RUP's SOA plug-in belonging to the *Analysis & Design* discipline and by other activities of the *Requirements* and *Analysis & Design* disciplines.	Part of design step.
Design of interfaces, messages structure and format		-do-
Model of services interoperability and cooperation		Formally part of design step, partly related to the integration.
Security model	Covered by *Analysis & Design* discipline.	Securing individual services.
Implementation of	Supported by activities of the	Generation of particular service

[4] The „RUP for SOA" column lists disciplines of the RUP methodology enhanced with specific SOA modeling plug-in that are related to the acitvities and/or deliverables of a service-based application life-cycle.

[5] The „XP" column briefly describes how XP might support individual activities of a service-based application life-cycle.

[6] The first version should contain only those jobs that "force" us to set up a skeleton of a whole application.

Activity/deliverable	RUP for SOA[4]	XP[5]
new services incl. description, semantics	*Implementation* discipline.	description documents can be automated by appropriate tools.
Reuse and/or localization of existing services		Reuse of existing services and their integration into an application.
Refactoring existing services		Can take place within periodic revisions.
Assignment of services and components into appropriate places within SOA.	Part of the *Identify Service* activity.	No explicit support.
Functional tests	Tests are implemented within the *Implementation* discipline activities and processed within activities of the *Test* discipline.	Tests are written before core functionality is implemented.
Integration tests		Run whenever new addition is available.
Availability and performance testing		*Testing* activity.
Application assembly	Performed within integration activities of the *Implementation* discipline.	Continuous integration (end of a day integration tests).
Deployment of the services into operating environment	Supported by activities of the *Deployment* discipline.	If version is finished and integration tests passed; data migration; operation tests.
Service publishing		Might be run as a part of final end-of-the-day work upload.
Operation monitoring	The RUP methodology life-cycle ends with the *Transition* phase and there are no further guidelines related to the product operations.	Yes, supported.
Maintaining problem logs		Implementation made ready for automated recording.
Detecting, localizing and solving failures		Hot-line, Help desks etc.
Tuning the operational environment		Yes, supported.
Routine maintenance		Yes, supported.
Feedback evaluation and process improvements		Focused more on developed software than the business process improvements.

Table 2 - level of support application life cycle by XP and RUP for SOA

4 Conclusion

The XP methodology as a representative of agile methodologies is primarily focused on as soon as possible, relatively small and frequent, software deliveries. The core sequence of activities consists of

creating tests, implementing functionality, presentation to the customer and usage of feedback in the next iteration.

From the perspective of a service-based application development and operation, the methodology is suitable mainly for application types II and III (see 2.1) – adapters and single service applications. In those cases XP covers all (or almost all) life cycle activities and there is no need to support those activities related to composite applications, like process modeling, service integration and cooperation within a process, putting a service into SOA or publishing of a service.

In case of composite application, RUP for SOA plug-in guides developers through the development process by supporting all the life-cycle activities except "Operation & Maintenance" tasks. In addition, there is a special plug-in of RUP focused on eXtreme Programming, thanks to which RUP can turn to account XP advantages.

5 Bibliography

[1] Arsanjani, A.: *Service-oriented modeling and architecture*. IBM developerWorks, IBM, 2004. URL http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design1/.

[2] Beck, K.: *Extrémní programování*. Praha, Grada 2002. ISBN 80-247-0300-9.

[3] Benatallah, B., Casati, F., Nezhad, H. R. M. and Toumani F.: *Developing Adapters for Web Services Integration*. Advanced Information Systems Engineering, 17th International Conference, CAiSE 2005, Porto, June 13-17, 2005. URL http://www.hpl.hp.com/personal/Fabio_Casati/docs/Caise05-adapters.pdf

[4] High, R., Kinder, S. and Graham, S.: *IBM's SOA Foundation: An Architectural Introduction and Overview*. IBM, November 2005. URL http://www-128.ibm.com/developerworks/webservices/library/ws-soa-whitepaper/

[5] Channabasavaiah, K., Holley, K. and Tuggle, E.: *Migrating to a service-oriented architecture, Part 1*. IBM developerWorks, IBM, 2003. URL http://www-128.ibm.com/developerworks/webservices/library/ws-migratesoa/

[6] Johnston, S.: *Modeling Service-oriented solutions*. IBM Rational, IBM developerWorks. 2005. URL http://www-128.ibm.com/developerworks/rational/library/jul05/johnston/