Selection and Evolutionary Development of Software-Service Bundles: A Capability Based Method

Jānis Grabis[1(✉)] and Kurt Sandkuhl[2]

[1] Institute of Information Technology, Riga Technical University, Kalku 1, Riga, Latvia
grabis@rtu.lv
[2] Chair of Business Information Systems, University of Rostock, Albert-Einstein-Straße 22, Rostock, Germany
kurt.sandkuhl@uni-rostock.de

Abstract. Software-service bundles are combinations of software products and services offered by their vendors to clients. The clients select a combination of software product and associated service best suited to their specific circumstances. The paper proposes an information sharing based method helping clients to select the most appropriate combination or configuration and also supporting the continuous improvement of the solution in response to changing circumstances. The method utilizes principles of the Capability Driven Development to characterize performance objectives and contextual factors affecting delivery of a software-service bundle. Application of the method is demonstrated using an illustrative example of data processing.

Keywords: Software selection · Capability · Evolutionary development · Software-service bundle

1 Introduction

Software-service bundles are combinations of software products and services aimed at providing packaged offerings for specific applications. Software vendors provide these solutions to their clients. These services have different configurations with regards to functionality provided. Software product line engineering [1] investigates the problem of creating the solutions efficiently from the vendor perspective. Clients on the other hand are looking for the most feasible solution meeting their specific requirements. This problem is addressed in software selection research [2] though these investigations mainly take into account only information available to a client. However, vendors have a wealth of information about their software used by other clients [3]. That could include contextual information describing unique operating circumstances of the client as well as information about performance achieved by using specific configurations of the solutions. It is argued that vendors and clients can collaborate for finding the right configuration for every client on the basis of sharing historical context and performance data. This way every client would receive a configuration appropriate for its operating context as well as some estimates of expected performance of the solution. From a

J. Krogstie et al. (Eds.): CAiSE 2016, LNBIP 249, pp. 3–14, 2016.
DOI: 10.1007/978-3-319-39564-7_1

service science perspective, the colla-boration between vendor and client is considered as a precondition for successfully implementing the service part of software-service bundles (i.e. co-creation of value).

In order to enable aforementioned approach, a framework for defining contextual information and performance objectives is required. Recently, capability driven development (CDD) has been proposed as an approach for ensuring that solutions can be delivered in different contexts at the desired level of performance [4]. The approach presumes that rather than providing a simple business solution the vendor possesses certain capabilities and is able to provide such a capability to its clients facing different operating circumstances. It is a model based approach and encompasses three development phases: (1) capability design explicitly defines performance goals, context factors affecting capability delivery and context-dependent capability delivery solutions; (2) capability delivery phase concerns monitoring of context and performance data and adjusting the solution in response to changes in these data; and (3) feedback phase provides information for updating of the initial design.

The objective of this paper is to elaborate a CDD based method allowing colla-boration between vendor and client in selection of the right configuration of software- service bundles and continuous improvement of the selected configuration. It is assumed that a vendor has multiple clients. The clients share usage information about the software-service bundle. In case of preparing a bundle for a new client, this information is used to create a decision-making matrix for selecting an appropriate configuration for this new client. The new client usually starts with a minimum satisfactory configuration; performance of this configuration is continuously monitored and if necessary the client upgrades its configuration which here is referred to as evolutionary development in analogy to evolutionary software development [5].

The main contributions of the paper are: (1) combination of vendor and client perspectives in an information sharing based method for selection of software–service bundles; and (2) selection of software-service bundle as an interplay among contextual data and performance objectives (i.e., selection is made in a context-aware performance driven fashion). The rest of the paper is organized as follows. Section 2 provides overview of the method. Section 3 elaborates stages of the evolutionary development. The application example is provided in Sect. 4. Related work is reviewed in Sect. 5. Section 6 summarizes findings and future work.

2 Method Overview

The evolutionary development method for software-service bundles is based on the CDD approach and uses the capability model underlying the software-service bundle as a starting point for providing appropriate configurations to clients.

2.1 Problem Statement

The vendor offers its clients a software-service bundle *S*. The software-service bundle consists of a software product, know-how and supporting services ranging from

helpdesk to business process outsourcing. S is designed in a way to deliver desired performance in different contextual situations, i.e., the vendor possesses the capability of providing the software-service bundle.

S is provided in one of N configurations $O_i,\ldots,O_N$ and the configurations differ by their price (they are ordered ascendingly starting with the lowest costs configuration). Delivery of S depends on M context factors $C_1,\ldots,C_M$ and its performance is measured by L key performance indicators $K_1,\ldots,K_L$. Combinations of values of the context factors yield a context situation describing specific solution delivery circumstances. It is assumed that certain configurations provide better performance for specific context situations than other, i.e., they are better suited for these context situations. For instance, a configuration including an outsourcing service works better in the case of highly variable demand for troubleshooting services.

There are P clients using one of the configurations. It is assumed that existing clients have an incentive to share anonymized values of context factors and key performance indicators (KPI) during operations of the software-service bundle.

Two decision-making challenges are: (1) to select appropriate configuration for a new client; and (2) to upgrade configurations used by existing clients in the case of changing circumstance or unsatisfactory performance. In the former case, selection is performed by matching a context situation of the new client with context situations supported by the vendor of the software-service bundle. In the latter case, an existing client switches from one configuration to another to adapt to changing circumstances.

2.2 Evolutionary Development Process

The aforementioned challenges are addressed following an evolutionary development process (Fig. 1). A vendor uses the CDD approach [4] to develop a capability model. The model specifies capability delivery goals, delivery context and solutions (i.e. software-service bundles and their appropriate configuration) for capability delivery offered to clients (see Sect. 2.3 for further discussion). The capability model covers all configurations supported by the vendor. Relationships among context situations and configurations are described in a capability support matrix (CSM). The matrix indicates configurations suitable for a particular context situation. It is used by the vendor and clients to find appropriate solution for clients' needs. CSM is developed on the basis of historical data analysis or according to judgement of the vendor.

Upon engaging a new client, its typical context situation is assessed and the least expensive configuration supporting this context situation is selected. The selected configuration is provided to the client. It is used for capability delivery and delivery performance is monitored using the indicators defined in the capability model. If performance targets are not achieved or context values venture outside the defined context element range, the capability delivery solution is adjusted. Potential adjustments are: (1) selection of a more appropriate configuration; or (2) designing a new solution. The capability monitoring and adjustment are performed cyclically and the capability delivery solution evolves according to the business requirements of the client. The vendor accumulates capability delivery performance and context data from multiple clients and uses this information to update the capability delivery solution and validate CSM.

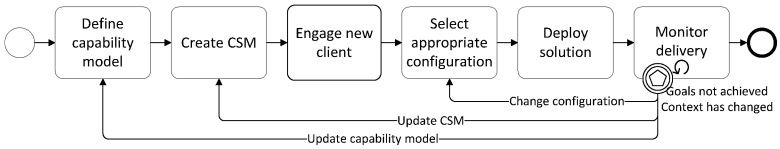

Fig. 1. The evolutionary capability development process

2.3 Capability Modeling

The capability model defines vendor's ability and capacity to provide a solution to clients facing specific circumstances. Figure 2 provides a simplified overview of the key elements used in capability modeling as well as their relation to the configuration concept used in this paper. Goals are business objectives the capability allows to achieve. They are measured by KPI. The capability is designed for delivery in a specific context as defined using context elements. The context elements name factors affecting the capability delivery while context situations refer to combinations of context element values. The process element specifies a capability delivery solution. Process variants describe the capability delivery process for a specific context situation while the associated configuration of the solution encompasses all technical, human and knowledge resources necessary to execute the process.

A configuration can include multiple process variants. The client can switch from one process variant to another or invoke them simultaneously during solution delivery depending on context situation.

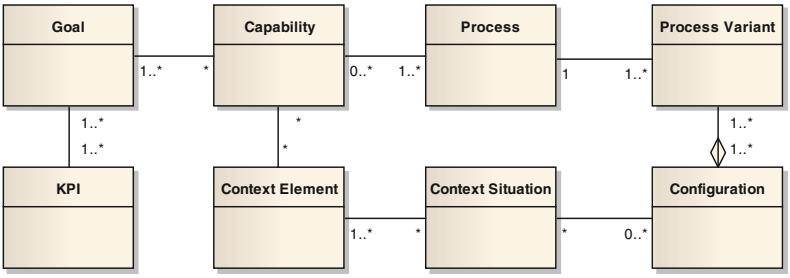

Fig. 2. Key concepts of capability modeling

From the evolutionary development perspective, the key aspects are that: (1) capability can be delivered in different context situations while each individual client faces just some of these context situations; (2) process variants specify a solution for dealing with one or several context situations. For software-service bundle these variants cover both, different process variants in the software product if this product has a process-oriented architecture and different variants for the service bundled with the software product as a part of configurations; and (3) relationships among context situations, performance and process variants are not necessarily know in advance and can be induced from the solution's usage data.

3 Evolutionary Development Stages

The evolutionary development process includes two distinctive phases: (1) design stage – when the initial configuration of the software-service bundle is selected and deployed for a new client; and (2) delivery stage – when the software-service bundle is used by the client and it is adjusted according to changing circumstances.

3.1 Design Stage

At the beginning of the design stage the vendor develops a capability model corresponding to the software-service bundle to be provided to clients. Every context factor used in selection of the configuration has a finite set of values or context range $CR_i = (cr_{i1}, \ldots, cr_{iT_i})$, where T_i represents a number of values for the ith context element. These values are obtained by categorizing actual values of context observations also referred as to measurable properties [6]. The categories enable for relative comparison of clients.

Combination of context element values form the context range yields a set of context situations $(CS_1, \ldots, CS_H) = CR_1 \times \ldots \times CR_N$ (H is the number of context situations). CSM is defined as a matrix with elements a_{ij}, $i = 1,\ldots,H$, $j = 1,\ldots,N$, where $a_{ij} \in \{0, 1\}$ relates context situations to suitable configurations. The matrix element $a_{ij} = 1$ indicates that configuration O_j is suitable in the case of context situation CS_i. The same configuration could be suitable for multiple context situations. One configuration could encompass multiple process variants.

Upon engaging a new client, its most plausible context situation is identified as CS_{new}. Appropriate solution is identified by

$$\min(j \mid a_{ij} = 1 \wedge CS^{new} \in CS_i) \tag{1}$$

Equation 1 selects the least cost configuration O_j appropriate for context situation faced by the new client. If no appropriate configuration is available, the client has a choice to select a configuration having the highest level of overlapping with support context situations. The client also sets target values for KPI $K_1^{new}, \ldots, K_L^{new}$, where the superscript refers to the new client and the subscript l refers to the KPI. The selected configuration is setup for the client and it is ready for operations. There could be a setup time for deploying the configuration.

3.2 Delivery Stage

During the delivery stage, actual context situations and delivery performance are monitored. The performance monitoring is carried out by gathering real-time values of KPI K_{it}^{new}, where superscript identifies the client, the subscript i refers to KPI being measured and t refers to the measurement time. The actual value is compared to the performance target. If $K_{it}^{new} < K_i^{new}$ then the ith performance objective is not met and a recommendation to revise the solution is issued. Obviously, one should evaluate to what extent the software solution is responsible for underperformance.

The context monitoring is performed by comparing the observed context situation CS_t^{new} for the new client at the tth time moment to the context situations supported by the current configuration, i.e., relationship $CS_t^{new} \in \mathbf{CS}_O$ where $\mathbf{CS}_O$ is a set of context situations supported by configuration O_j used by the client. If the relationship does not hold then a warning is issued notifying that the observed context situation is not explicitly supported by the current configuration. The context monitoring serves as an advanced warning system to potential performance deterioration since it is not known whether the current configuration is suitable for the observed context situation. That might lead to an unexpected behavior.

3.3 Evolution

Violations of performance objectives or observation of unsupported context situations triggers a warning suggesting an upgrade of the current configuration. In response, to this warning a client might decide on upgrading the current configuration by selecting a more suitable configuration from CSM. This is a suitable approach if the actual context situation is different from the one identified during the design stage or it has changed. However, if underperformance is observed for the supported context situation and it is attributed to the software product then the vendor might need to reevaluate CSM or a special software-service bundle needs to be developed for the particular client.

4 Application Example

Business processes in many industries require collaboration among two or more companies. Often this collaboration involves business information exchange in a form of data exchange messages [7]. Such messages can contain errors, which need to be corrected before further utilization of the information. The correction of errors might require manual interventions which will be referred to as “cleaning services”. The example is motivated by a real-life case in the energy industry, where a software development company provides software for exchanging energy consumption data as well as associated business process outsourcing services for handling data errors [8].

4.1 Description

A vendor offers a business information processing service. The service consists of data processing software, data processing services and – if required – cleaning services by the back office staff of the vendor based on knowledge about the most common data exchange exceptions. The data processing services ensure business information processing on behalf of the client. Clients can choose between doing data processing and cleaning in-house or outsourcing it to the vendor. Figure 3 shows an overall data exchange process from the client’s perspective. The client receives a message. Depending on a decision-making logic the messages are processed along one or several process branches. The first branch represents a manual processing (client’s employees correct errors). The second branch represents an automated processing using the

knowledge base on common exceptions provided by the vendor. However, some of the exceptions might require manual intervention ("cleaning"). A client uses the outsourcing service provided by the vendor to deal with exceptions in the third branch. The client transfers the exceptions to the outsourcing service and receives back the remedied data. Multiple branches can be used simultaneously. For example, the client mainly uses in-house automated processing and invokes the outsourcing service only if internal resources are overloaded. This decision is made during the service delivery. Nevertheless, the solution should be configured in a way to support both automated in-house processing and usage of the outsourcing service.

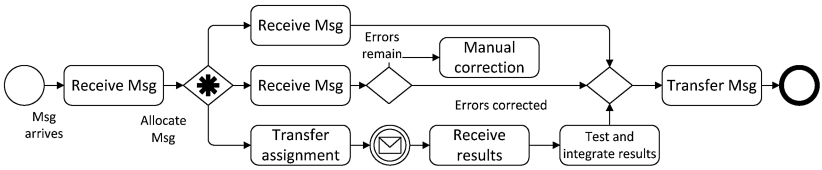

Fig. 3. The overall business information exchange process.

The process execution goals are timely processing of all messages and handling of all exceptions. The main context factors affecting the process execution are the number of data exchange messages received or processing load and load volatility. The load volatility characterizes variations in the processing load what might have adverse consequences on scheduling of resources assigned to the manual processing.

4.2 Model

The vendor possesses the data exchange capability provided to its clients by means of the software-service bundle. The data exchange capability model is created using the concepts defined in Sect. 2.3. Goals, context and process variants are the main elements of the capability model important for the software solution selection method.

The main data exchange goals are timely data processing, correction of data exchange errors as well as cost minimization and efficient utilization of resources involved in the data exchange process. These goals are measured by the corresponding KPIs. For instance, the timely data processing goal is measured as the processing time KPI K_{PT}. The context elements affecting the capability are defined in Table 1. The processing load context element is measured by the number of messages received per day and it assumes values from the range of values. Not all context elements are used in configuration selection for the software-service bundle. Current backlog and schedule context elements are used for run-time decision-making.

The capability model also defines the overall data exchange process including three processing variants (Fig. 3). These process variants serve as the basis for defining configurations of the software solution. The Allocate messages gateway represents decision logics for run-time allocation of messages among process variants if several of them are included in the configuration.

Table 1. Context values and their context ranges

Context element	Context element range
Processing load (C_{PL})	Low, medium, high
Processing load volatility (C_{LV})	Low, medium, high
Backlog (number of messages waiting for processing)	0…1000
Calendar (scheduled hours for human resources)	0…100

Table 2. Capability support matrix for data exchange software-service bundle

Processing load level	Load volatility	O_1	O_2	O_3
Low	Low	1		
Low	Medium	1	1	
Low	High		1	
Medium	Low		1	
Medium	Medium		1	1
Medium	High			1
High	Low		1	
High	Medium			1
High	High			1

Three configurations are offered to clients: O_1 – manual processing of data exchange exceptions; O_2 – automated processing of data exchange exceptions; and O_3 – combination of automated processing of data exchange exceptions with availability of exceptions handling outsourcing services. The vendor also uses its expertise and historical data to prepare CSM (Table 2). The matrix lists context situations as combinations of values of C_{PL} and C_{LV} context elements. It shows that, for instance, configuration O_1 is suited for CS_1 = {low, low}. Advanced configurations could be used in simple context situations though that is not promoted to avoid inefficiencies.

4.3 Results

The aforementioned capability model provides foundation for delivering data exchange solutions to clients. A simulated experiment is conducted to illustrate the software-service bundle selection method. It simulates a flow of data exchange messages for a single client and the client attempts to process these messages using one of the solutions provided by the vendor. Execution of manual data processing activities in all configurations requires human resources drawn from a limited pool of resources and has a variable duration depending on complexity of exceptions.

The message flow D_t varies over time and is described as an autoregressive process $D_t = \rho + \alpha D_{t-1} + \varepsilon$, where ρ and α are coefficients defining process shape and $\varepsilon = N(0, \sigma)$ is normally distributed with the standard deviation σ. The average flow of messages $\mu = \rho/(1-\alpha)$ and affects the processing load context element. The relationship between D_t and value of the processing load context element C_{PL} is expressed as

$$C_{PL} = \begin{cases} \text{low, if } \mu < 100 \\ \text{medium, if } 100 \leq \mu < 1000 \\ \text{high, if } \mu \geq 1000 \end{cases} \quad (2)$$

The coefficients α and σ affect processing load volatility, i.e., larger values of these coefficients result in a more volatile message flow. In this experiment $\alpha = 0.8$ and $\sigma = \mu/5$. The value of $\rho$is varied to evaluate different context situations: (1) In the first experiment (EXP1) ρ is set to 10 to evaluate a low processing load situation; and (2) in the second experiment (EXP2) ρ is increased from 10 to 100 during the course of message processing simulation to evaluate the impact of changes in context. Numerical values used in the experiments are practically grounded though do not represent actual observations. A new client defines that its typical context situation CS^{new} = {low, low}. The least cost configuration appropriate for this context situation is O_1. This configuration is setup for the client. That implies client receiving data exchange software and using manual exceptions handling.

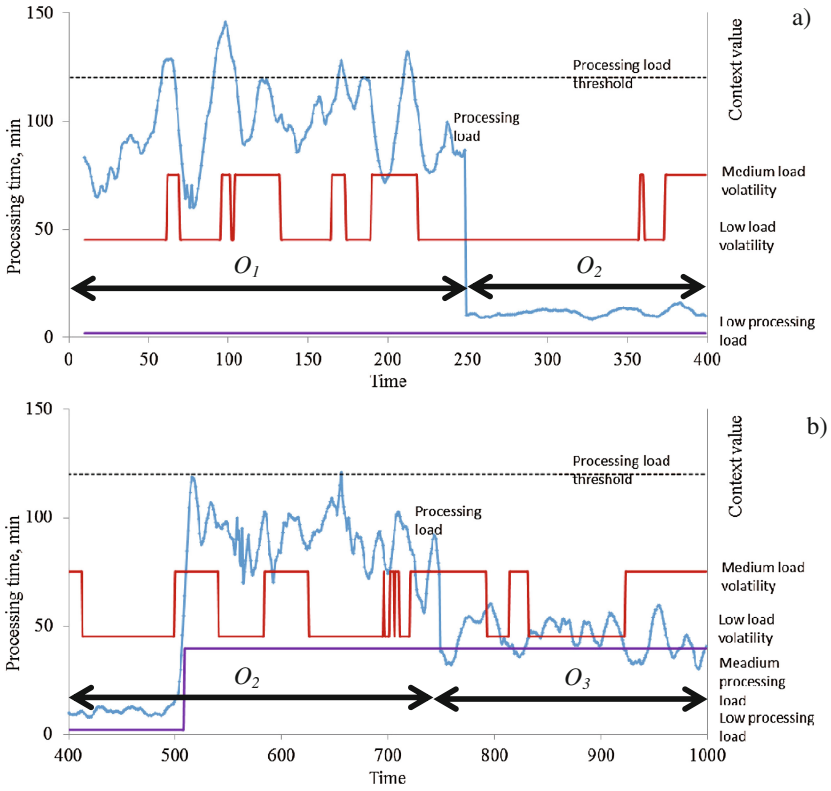

Fig. 4. Dynamics of simulated delivery results and configurations used: (a) EXP1 and (b) EXP2

Figure 4a shows monitoring results for EXP1. It includes values of $K^{new}_{PT,t}$ and the threshold value $K^{new}_{PT,t}$ = 120 min. The processing load context element C_{PL} is constant while load volatility C_{LV} exhibits slight variations and occasionally assumes Medium value not explicitly supported by the current configuration O_1. More importantly, $K^{new}_{PT,t}$ frequently exceeds the threshold value what triggers a recommendation to reconsider the configuration. CSM suggests that O_2 is appropriate for dealing with CS_2 = {low, medium}. The switch to O_2 takes place at time period 250. One can observe that performance is significantly improved.

EXP2 simulates a permanent change of the context situation and C_{PL} assumes Medium value. O_2 is suitable for both CS_4 = {medium, low}and CS_5 = {medium, medium}, and experimental results (Fig. 4b) show that O_2 delivers satisfactory performance after the change in context. However, $K^{new}_{PT,t}$ is close to its threshold. Assuming, that a similar behavior is observed also for other clients using O_2 in similar conditions, the vendor might decide on updating CSM and recommending to used exclusively O_3 in context situation CS_5 = {medium, medium}. The simulated switching takes place at time period 750, when O_3 is deployed. This change results into reduction of the processing time.

The experimental results demonstrate that context observations can be used to drive selection of appropriate configurations on the basis of the common capability model.

5 Related Work

Related work can be found in the areas of selection of packaged applications and version management. The multi-objective methods for selecting packaged applications approach allows for comprehensive evaluation of offerings by different vendors [2]. The selection is based on the generic set criteria and the alignment of these criteria with business needs is not ensured. Capilla et al. [9] describe methods for designing different versions of business software depending on the application context. These works focus on software architecture and design issues rather than the software management decisions. Evolutionary development in software engineering [5] concerns the operational level evolution of atomic development requirements.

The method proposed relies on several existing methods. Goal modeling techniques [10] help to identify the relevant objectives. Business activity monitoring techniques allow measuring the process performance according to the specified goals and to identify significant changes in performance [11]. Context model techniques [12] help to identify relevant context factors and to represent their impact on process design. The method extensively uses categorized context values. A similar approach is taken by [13] to model context-driven business processes.

Furthermore, there is related work in the field of service management from a service science perspective. In recent years, the perspective on what is characterizing a service has shifted from an intangible product to a more process-oriented focus [14]. A service process "[…] can be viewed as a chain or constellation of activities that allow the service to function effectively" [15, p. 68]. Existing work tries to understand service processes from three overall perspectives: input, transformation process and outcome

[16]. In contrast to manufacturing-based production processes, also customers provide significant input in service processes [17]. This is clearly visible in our product-service bundle. However, this input is not only limited to one customer, but also multiple customers which is also acknowledged service science [18]. The transformation process "entails the service delivery and consumption process, and involves customer participation in the service delivery/consumption process" [16, p. 1016]. For software-service bundles, we thus can divide this perspective into the continuum of service co-production [19] and consumption process flow. Latter is considered in existing work as a characteristic of services from a service operations management perspective. The final outcome of the service is determined by the service provider as well as by the service beneficiaries [20].

6 Summary and Future Work

The paper presented and discussed a method for evolutionary development and configuration of software-services bundles. The paper showed that the vendor-client collaboration for selecting the most suitable configuration of a given software-service bundle is feasible and represents value co-creation between vendor and client. The capability concept and the CDD approach have been useful for this method to provide the common basis for defining software solution and explicitly representing relationships among performance, context and solutions. The method so far is evaluated only using a simulation approach and experiences from real-world application cases are needed for further validation.

There are multiple directions of further research. Updating the capability support matrix according to monitoring results is an important part of the method what requires further elaboration. Classification and machine learning methods can be used for these purposes. The decision to upgrade the solution is not an automated decision and usually involves a number of considerations (e.g., business relations) not covered by the evolutionary development method. Switching to a new configuration incurs additional costs. This factor also could be incorporated into the decision-making process to evaluate cost-benefit aspects currently not considered in the paper.

From a service science perspective, the co-production of services has to be seen as a continuum which "[…] can vary from none at all to extensive co-production activities by the customer or user" [21, p. 8]. When specifying service processes, it thus needs to be highlighted which tasks are performed by which entity of the service system or service system network. For software-service bundles, modeling of the service process with explicit distribution of tasks between vendor and client could be a way to further optimize gathering of relevant data. This will be part of future work.

References

1. Pohl, K., Böckle, G., Van der Linden, F.: Software Product Line Engineering. Foundations, Principles, and Techniques. Springer, Heidelberg (2005)
2. Jadhav, A.S., Sonar, R.M.: Evaluating and selecting software packages: a review. Inf. Softw. Technol. **51**, 555–563 (2009)

3. Olsson, H.H., Bosch, J.: Towards continuous customer validation: a conceptual model for combining qualitative customer feedback with quantitative customer observation. In: Fernandes, J.M., Machado, R.J., Wnuk, K. (eds.). LNBIP, vol. 210, pp. 154–166. Springer, Heidelberg (2015)
4. Bērziša, S., Bravos, G., González, T., Czubayko, U., España, S., et al.: Capability driven development: an approach to designing digital enterprises. Bus. Inf. Syst. Eng. **57**, 15–25 (2015)
5. Sommerville, I.: Software Engineering. Pearson, Boston (2015)
6. Grabis, J., Stirna, J.: Advanced context processing for business process execution adjustment. In: Persson, A., Stirna, J. (eds.) CAiSE 2015 Workshops. LNBIP, vol. 215, pp. 15–26. Springer, Heidelberg (2015)
7. Schmidt, A., Otto, B., Österle, H.: Integrating information systems: case studies on current challenges. Electron. Markets **20**, 161–174 (2010)
8. Sandkuhl, K., Koc, H.: On the applicability of concepts from variability modelling in capability modelling: experiences from a case in business process outsourcing. In: Iliadis, L., Papazoglou, M., Pohl, K. (eds.) CAiSE Workshops 2014. LNBIP, vol. 178, pp. 65–76. Springer, Heidelberg (2014)
9. Capilla, R., Ortiz, O., Hinchey, M.: Context variability for context-aware systems. Computer **47**, 85–87 (2014)
10. Kavakli, E.: Modeling organizational goals: analysis of current methods. In: Proceedings of the ACM Symposium on Applied Computing, pp. 1339–1343 (2004)
11. Friedenstab, J., Janiesch, C., Matzner, M. Müller, O.: Extending BPMN for business activity monitoring. In: Proceedings of the Annual Hawaii International Conference on System Sciences, pp. 4158–4167 (2012)
12. Koç, H., Hennig, E., Jastram, S., Starke, C.: State of the art in context modelling – a systematic literature review. In: Iliadis, L., Papazoglou, M., Pohl, K. (eds.) CAiSE Workshops 2014. LNBIP, vol. 178, pp. 53–64. Springer, Heidelberg (2014)
13. Born, M., Kirchner, J., Muller, J.P.: Context-driven business process modeling. In: Camp, H. S.O. (eds.) Advanced Technologies and Techniques for Enterprise Information Systems, ICEIS 2009, pp. 17–26 (2009)
14. Sampson, S.E.: Visualizing service operations. J. Serv. Res. **15**, 182–198 (2012)
15. Bitner, M.J., Ostrom, A.L., Morgan, F.N.: Service blueprinting: a practical technique for service innovation. Calif. Manage. Rev. **50**, 66–94 (2008)
16. Yalley, A.A., Sekhon, H.S.: Service production process: implications for service productivity. Int. J. Prod. Perform. Manage. **63**, 1012–1030 (2014)
17. Sampson, S.E., Froehle, C.M.: Foundations and implications of a proposed unified services theory. Prod. Oper. Manage. **15**, 329–343 (2006)
18. Tax, S.S., McCutcheon, D., Wilkinson, I.F.: The service delivery network (SDN) a customer-centric perspective of the customer journey. J. Serv. Res. **16**, 454–470 (2013)
19. Hilton, T., Hughes, T.: Co-production and self-service: the application of service-dominant logic. J. Mark. Manage. **29**, 861–881 (2013)
20. Spohrer, J., Kwan, S.K.: Service science, management, engineering, and design (SSMED): an emerging discipline. Int. J. Inf. Syst. Serv. **1**, 1–31 (2009)
21. Vargo, S.L., Lusch, R.F.: Service-dominant logic: continuing the evolution. J. Acad. Mark. Sci. **36**, 1–10 (2007)