Software Architecture Evaluation in Global Software Development Projects

Frank Salger

Capgemini sd&m, Carl-Wery-Straße 42, 81739 Munich, Germany
frank.salger@capgemini-sdm.com

Abstract. Due to ever increasing system complexity, comprehensive methods for software architecture evaluation become more and more important. This is further stressed in global software development (GSD), where the software architecture acts as a central knowledge and coordination mechanism. However, existing methods for architecture evaluation do not take characteristics of GSD into account. In this paper we discuss what aspects are specific for architecture evaluations in GSD. Our experiences from GSD projects at Capgemini sd&m indicate, that architecture evaluations differ in how rigorously one has to assess modularization, architecturally relevant processes, knowledge transfer and process alignment. From our project experiences, we derive nine good practices, the compliance to which should be checked in architecture evaluations in GSD. As an example, we discuss how far the standard architecture evaluation method used at Capgemini sd&m already considers the GSD-specific good practices, and outline what extensions are necessary to achieve a comprehensive architecture evaluation framework for GSD.

1 Introduction

In software development projects, the software architecture constitutes the "glue" between requirements, design and, ultimately, the code base. It determines whether the nonfunctional requirements will be met, and whether the code base will eventually embody a homogenous texture.

In global software development (GSD), high quality software architectures play an even more important role than in co-located development projects: Architectural structure determines organizational structure and vice versa [1, 2]. It is more difficult to enforce compliance to architectural rules, and to have them "sink in" [3, 4]. Finally, unplanned and informal contact between team members where architecturally important information is casually exchanged does not take place between remote teams in GSD [5]. Hence, some aspects of architecting a software system seem to be different in GSD as compared to co-located software development.

Today, powerful and mature methods to evaluate software architectures are at hand (see e.g. [6, 7], and [8] for an overview). Comprehensive guidance on how to conduct architecture evaluations is available [9, 10, 11].

However, despite the fact that architectural work is considerable different in GSD, current approaches to architecture evaluation do not answer the important question: "How should architecture evaluations differ when applied in the context of GSD?"

R. Meersman, P. Herrero, and T. Dillon (Eds.): OTM 2009 Workshops, LNCS 5872, pp. 391–400, 2009.

With this paper we try to give some first answers to the above question and further stimulate the discussion what aspects are specific for architecture evaluations in GSD.

In the next section, we present experiences from various GSD projects at Cap gemini sd&m and compare them to existing research literature. We then derive nine architecture good practices, which we believe should be checked in an architecture evaluation method used in GSD. In section 3, we describe the standard method to evaluate software architectures at Capgemini sd&m. We explain some shortcomings of this method when applied in GSD settings, and discuss how it can be improved in order to increase its effectiveness for GSD settings. We draw our conclusion in section 4 and outline our future research agenda.

2 Hot Spots for Architecture Evaluation in GSD Practice

What are the pressing architecture related problems in GSD in the industrial practice? To approach this question, we discuss the lessons learned gained at Capgemini sd&m from numerous GSD projects for custom development of large scale business information systems. We match our experiences to results from the GSD literature, thereby confirming both, our experiences as well as the respective research results.

To structure the rest of the paper, we observe that the word "architecture" is often used to describe three aspects of architectural work [6]: the organization of a system (the '*product*'), the description of this organization (the '*documentation*'), and finally the process of elaborating these descriptions (the '*process*'). We will use these three clusters when we subsequently discuss our project experiences and compile a list of our architecture good practices for GSD.

2.1 Lessons Learned from GSD Projects at Capgemini sd&m

Capgemini sd&m is the Technology Services unit of the Capgemini Group in Germany and Switzerland and offers its customers end-to-end process and software solutions that have a crucial impact on their competitiveness. The Capgemini Group runs captive centers for offshoring software development in several countries, including India. We now discuss the some experiences distilled from various GSD projects carried out at Capgemini sd&m. We here solely focus on projects for custom software development of business information systems. When we use the acronym 'GSD' in the following, we always refer to this kind of projects.

2.1.1 Product

Issue 1: In one of the biggest GSD projects at Capgemini sd&m (encompassing several hundred use cases), the modules where defined too coarse grained: They bundled the functionality of several dozen use cases not all of which where closely related to each other from a functional point of view. Some modules where too big to be handled by one team. Time pressure forced to assign dislocated teams to work on the same modules. These teams experienced a lot of extra communication: They had to repeatedly sort out, which team had to develop which functionality. Time pressure further decreased communication, which finally let to an increased amount of code duplicates. We thus affirm existing results that one of the most challenging product

related issues is to find the right modularization supporting the communication of distributed teams [1, 2, 5, 12]. This is a very intriguing issue to resolve. A naïve approach would be to minimize all communication between teams that reside at different shores. But this increases the risk of isolating the distributed teams thereby decreasing mutual awareness of remote team members for each others working status.

Issue 2: This project also suffered from a mismatch between the use cases and the business processes of the customer. This became obvious too late in the project and let to considerable rework of the use case model. We experienced that such structural changes are more difficult to deal with in GSD than in co-located settings.

Issue 3: Another recurring problem is the instability of the architecturally significant decisions which often result from instable or overlooked non-functional requirements. In this project, the technical proof-of-concept was insufficient. As a consequence, some of the core frameworks had to be repeatedly adapted after development team ramp up. This led to considerable unplanned extra effort to adapt the rest of the code base to these structural changes. Starting with a firm technical basis and stable non-functional requirements is mandatory in GSD. This matches the observations in [2, 13].

2.1.2 Documentation

Issue 4: One core learning from GSD projects at Capgemini sd&m is the need for comprehensive documentation to compensate communication and coordination issues. We observed however, that a GSD project does not necessarily need to document *more* than a co-located project. Often it is necessary to change *how* to document. As one example, not all software architecture documents written by German architects at Capgemini sd&m adhere to the 4+1 view format for architectures as suggested by the Rational Unified Process (RUP). But RUP is usually employed as main process model at Capgemini's captive centers in India. Therefore it is crucial to agree on a format for the architecture document which supports all development teams doing their work. We also experienced the importance of the right balance between codification and personalization of architectural knowledge (see also [14]). In our GSD projects we complement concise architecture documentation with intensive coaching.

Issue 5: Another stumbling point we encountered in our GSD projects is the problem of translation. Although the software engineers are well educated, it is sometimes challenging to write a software specification or an architecture document in a foreign language. Programming languages, architectural styles or design pattern constitute to some extend a universal language to software engineers. But complex non-functional requirements are already hard to capture precisely in the native language, but much harder in a foreign one. The same holds for specific terminology of less standardized and still evolving business domains. It is difficult to systematically translate a couple of thousand pages consistently. We sometimes experienced that terms in figures where translated differently to the according terms within the text. Also, short names of entities where not translated analogous to the long names of the entities. Usually, it was necessary to review the every translation.

2.1.3 Process

Issue 6: In one of our early GSD projects, we did not employ tools and processes to rigorously manage and control architecture evolution and prohibit architecture

erosion. This ultimately let to many headaches at the integration points: only there it became obvious that many architecture violations existed, which entailed laborious rework and refactoring phases. These unplanned efforts increased time pressure, which in turn further decreased the willingness to control architecture evolution. We also experienced that compliance to rules and processes was difficult to enforce. This matches the results in [3, 4].

Issue 7: These problems were further intensified by an inappropriate integration strategy: Formal code integration was done only on a monthly basis. This repeatedly lead to the undesirable big bang effect, where it became obvious that large extends of the code did not fit to each other. We thus confirm existing literature which stresses the importance of regularly and often integrating the whole code base [13, 15].

Issue 8: At the beginning, this project was planned as pure onsite, co-located development project. Offshore development had been "planned in" only after the project was well underway. The project then suffered from the fact that both, the business analyst and the technical architect joined the team much too late. It was very hard for the offshore team to catch up the same level of technical and business knowledge as the onsite team. We learned that collocating the onsite and offshore architects during the high-level architecture phase is crucial. The offshore architects can than act as 'architecture evangelists' when they are back in their home office. These experiences coincide with known good practices for architectural knowledge management [14].

2.2 Architecture Good Practices

From the discussion above, we can now derive our architecture good practices for GSD. These serve as an input for the subsequent discussion, what aspects should be assessed in architecture evaluations in GSD.

2.2.1 Product

GP1: *Ensure that the architecture fits to the organizational structure.* Establish a product architecture that minimizes communication of distributed teams but at the same time avoids communication breakdown. This addresses **issue 1**.

GP2: *Ensure that the architecture is aligned to business drivers.* Make sure that the architecturally significant decisions satisfy the architecturally significant requirements (ASR). Ensure that the alignment of the ASR to the relevant business drivers has been verified by the business analyst. This avoids potentially disastrous additional effort to adapt the architecture which becomes necessary if core ASR are changed. This good practice helps to mitigate **issue 2**.

GP3: *Ensure that the architecture is stable.* Assess the maturity and stability of the high-level architecture before programming starts to avoid that structural problems surface after lots of low level code has been written. This mitigates **issue 3**.

2.2.2 Documentation

GP4: *Consider the developers point of view.* Write the architecture document such that the offshore development team understands it and control that the team can actually work with it. Take the differing architectural approaches, methods, terminology and background of distributed teams into account. This alleviates **issue 4**.

GP5: *Check all translations.* Ensure that translations of the architecture are correct to avoid ambiguities and repeated rounds of clarification. This helps to avoid **issue 5**.

2.2.3 Process

GP6: *Co-locate architects during high-level architecture activity.* Co-locate offshore and onsite architects during the high-level architecture phase to efficiently disseminate core architectural knowledge and foster a common view on the system under construction. This good practice helps to avoid **issue 8**.

GP7: *Implement ongoing architecture management.* Make sure that architecture management processes are in place (and in use), to avoid architecture erosion. Make use of automatic tools to check for architecture compliance. This mitigates **issues 6**, **7**.

GP8: *Invest in architecture knowledge transfer and control it.* Knowledge transfer is more difficult in GSD that in co-located development [3]. For example it is known, that in Eastern countries, power distance is higher than in Western countries [15]. This can, for example, lead to situations, where the offshore developer does not pinpoint a flaw in the architecture to avoid that the onsite architect 'looses his face'. We experienced however, that such effects quickly fade out, as soon as the 'one-team-spirit' kicks in. This avoids **issue 8**.

GP9: *Enforce clearly defined configuration management policies.* Invest in clear policies for configuration-, build- and integration-management. Keep the knowledge about all involved processes up to date and control strict adherence to processes and policies. This mitigates **issue 7**.

From these nine good practices, GP 4, 5, 6 (and to some extend 8) are specific for GSD. The importance of the others however is usually strongly amplified in GSD, as the literature exemplifies [13]. We believe that assessing compliance to the above listed good practices is important and should be done in any architecture evaluation method applied at GSD of business information systems. These issues might not be the only ones which must be assessed. But we think that they lead the focus on issues with a potentially disastrous impact on project success, if they are not carefully addressed. In the next section, we describe the 'architecture quality gate'. We discuss which architecture related problems of GSD are not yet addressed by this method.

3 The Architecture Quality Gate Used at Capgemini sd&m

One of the business areas of Capgemini sd&m is the development of information systems for business-critical processes of our customers. Soundness, appropriateness and maintainability of our software architectures are major success factor of such projects. We thus devised a comprehensive architecture evaluation method which is used throughout Capgemini sd&m, the so called 'architecture quality gate' (QG-Arch) [17]. Figure 1 depicts the embedding of the quality gates within the overall quality assurance framework of Capgemini sd&m. It shows how the quality gates are complemented with ongoing quality assurance processes, like testing.

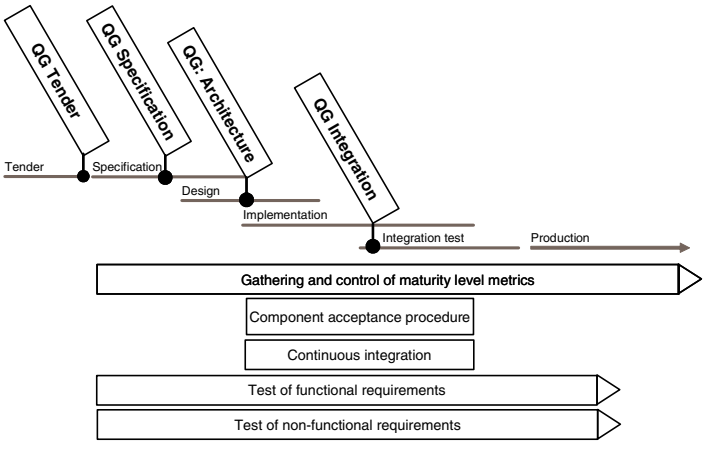

Fig. 1. Quality gates in context [17]

The high-level objectives of the QG-Arch are:

- To evaluate, whether the customer needs are met by the software architecture.
- To determine, whether the architecture is an appropriate solution for the problem.
- To secure, that the architecture is viable for the following development activities.
- To enforce the defined fundament for the following development activities.

The QG-Arch consists of six evaluation steps. In the first step, the software requirements specification is checked whether (1) functional requirements be understood by the developers, and (2) the specification was continued in the way, it was approved by our 'specification quality gate' (see [18] for details on this gate). In the second step, we apply a multi-level checklist in order to get an overview on the overall architecture. This serves as an input for the third step, where we apply 'lightweight-ATAM', an adaptation of the 'Architecture Tradeoff Analysis Method' [6] specifically tailored to our purposes. There, we verify the architecturally significant decisions against the quality model of the relevant stakeholders. In the fourth step we check, whether the evolutionary prototype is sufficiently mature to serve as a template (for example, compliance to coding style, robustness). Further, we check whether the behaviour of the system to be built can really be inferred from the exploratory prototypes. The fifth step serves to generate an explicit list of the main architectural problems together with a detailed investigation of these problems. In the last step, we assess the employed project management and the quality management with respect to architectural concerns. All these steps are supported by comprehensive usage concepts, guidelines, checklists and templates. The effectiveness of these steps has been proved in several large scale projects [17].

In the following, we describe how the QG-Arch addresses the core issues as listed in the last section, as well as the weaknesses of the QG-Arch with respect to GSD. We state in brackets which good practices are addressed by the QG-Arch, and which are not.

3.1 Product-Related Evaluation

Current practice in QG-Arch: Capgemini sd&m established a life-cycle encompassing method framework for crafting highly cohesive, loosely coupled architectures, which at the same time helps to avoid over-engineering. The framework consists of constructive and analytical methods which keep the business drivers of the customer aligned with the resulting system architectures throughout the development process: Starting at the enterprise level, we use Quasar Enterprise to build true service oriented architectures which support the customers' business processes and keep the application landscape architecture flexible [19]. At the "one-system"-level, we apply Quasar [20] to create flexible systems composed out of cohesive and loosely coupled modules. We check compliance to Quasar in the QG-Arch (**GP1 partly covered**). Alignment of the architecturally significant decisions to the architecturally significant requirements is checked with the 'lightweight-ATAM' [17] (**GP2 covered**). We also check, that the architecture is reasonable stable which the project usually proves by developing a proof of concept or prototype (**GP3 covered**).

Loose ends of QG-Arch with respect to GSD: In GSD, the right approach is to minimize the kind of 'unwanted' communication induced by highly dependent components by carefully crafting highly cohesive and loosely coupled components. But at the same time we seek to intensify the 'precious' communication about the overall system goals and context in order to repeatedly align the common view on the system under construction. Literature exists on distribution models (see e.g. [21]), but little is known about measuring how well the organisation fits the architecture [12]. Though Quasar helps us to build cohesive and loosely coupled architectures and QG-Arch checks compliance to Quasar, we believe that GP1 is not yet fully addressed.

3.2 Documentation-Related Check

Current practice in QG-Arch: The lightweight-ATAM checks whether architecturally significant requirements are explicitly documented (see [17]).

Loose ends of QG-Arch with respect to GSD: Assessing the architecture documentation as such is only insufficiently captured in the current version of the QG-Arch. The focus clearly lies on checking the appropriateness of the architecturally relevant decisions with respect to the architecturally relevant requirements. The QG-Arch does not explicitly assess readability and usability of the architecture document from the viewpoint of the developer. This however would be especially important in GSD, due to differing terminologies between the onsite and the offshore team. As the QG-Arch was mainly designed for the assessment of pure onsite projects, potential translation issues where also not taken into account. (**GP 4, 5 not covered**)

3.3 Process-Related Check

Current practice in QG-Arch: Here, we check whether precise quality objectives are defined for architecture artifacts, and how they will be monitored. How requirements will be traced back, how configuration and change management is implemented, how

architecturally related risks are determined and tracked, and how architecture management is implemented in order to cope with an evolving architecture. We also check that architecturally related milestones are defined, progress for architecture development is controlled and that all key roles are staffed with the right persons.
Loose ends of QG-Arch with respect to GSD: In co-located projects, it is common that the architects have a weekly meeting together with the business analyst and the project manager. We do not check explicitly in the QG-Arch that such meetings are carried out. (**GP 6 not covered**)

Although the use of automatic architecture management tools is by now commodity at Capgemini sd&m projects [17], we still have to investigate in which way configurations of these tools could be different for GSD as opposed to pure onsite projects. There are indicators, that hampered coordination and communication might mandate the use of smaller modules than usual. For the same reason it might be necessary to use stricter configurations for automatic tools, like code duplicate checkers. (**GP 7 covered**)

Knowledge transfer is not explicitly checked in the QG-Arch, since all Capgemini sd&m employees are repeatedly trained in the Capgemini sd&m methodologies like Quasar Enterprise [19], Quasar [20], or the Specification Methodology [18]. Although we carry out intensive trainings for our offshore colleagues, this cannot provide an equally strong common ground. We thus have to augment the QG-Arch by checks assuring an ongoing knowledge transfer as needed in GSD (**GP 8 not covered**).

Although we have standard policies for configuration, version, build and integration management, we do currently not comprehensively check compliance to them in the QG-Arch (**GP 9 partly covered**).

In summary, we found that the QG-Arch is already provides a good fundament which can be extended to an architecture evaluation method for GSD, although it does not yet cover the good practices GP 4, 5, 6 and 8. This is not surprising since it was originally devised for the use in multi-site onshore projects. However, moving to global multi-site software development introduces new challenges like communication and coordination problems, language problems and cultural differences [3].

Apart from extending the QG-Arch such that it also covers the missing good practices, we have discovered another topic for future work: The main goal we want to achieve with the QG-Arch is to avoid major problems with the software architecture [17]. As such, the QG-Arch was not designed to assess the lower level design within modules which however is important, since modules are often used as ‘work packages’ which can be handed over between distributed development teams. It is well known that such ‘handover checkpoints’ are an important means to manage the distributed development process [16]. We will thus devise a handover checkpoint, where the completeness of work packages and their compliance to previously agreed quality objectives is assessed jointly by the onsite and offshore team.

Within the QG-Arch, we also carefully assess architecturally relevant processes. This can be seen as an advantage over existing architecture evaluation methods like ATAM or SAAM [6], which purely concentrate on assessing the ‘product’- and ‘documentation’-aspects of software architectures. It is well known that assessing architecture-relevant processes is especially important in GSD [3, 4].

4 Conclusion

In this paper, we investigated how architecture evaluation methods should be adapted in order to take the characteristics of global software development (GSD) into account. To this end, we first discussed typical problems which we encountered in large scale commercial GSD projects at Capgemini sd&m. From this discussion, we distilled a list of good practices for GSD of large business information systems.

Product:

- Ensure that the architecture fits to the organizational structure
- Ensure that architecture is aligned to the business drivers
- Ensure that the architecture is stable

Documentation:

- Consider the developers point of view
- Check all translations

Process:

- Co-locate architects during high-level architecture activity
- Implement in ongoing architecture management
- Invest in architecture knowledge transfer and control it
- Enforce clearly defined configuration management policies

We argued that a software architecture evaluation method for GSD should check compliance to these good practices.

We then investigated to what extend the standard architecture evaluation method of Capgemini sd&m (the 'architecture quality gate', QG-Arch for short) already takes these results into account. We identified some shortcomings of the QG-Arch with respect to its effectiveness in GSD. Finally we discussed how to fill these loopholes of the QG-Arch. We also motivated that the QG-Arch should be complemented by handover checkpoints, where the internal design of modules is assessed, and which are jointly applied by the offshore and onsite architects.

Our main conclusions are that architecture evaluations which where devised for co-located project settings must be extended in two ways: 1) Architecture documentation must be carefully evaluated from an offshore developer point of view, and 2) the effectiveness of processes like architecture knowledge transfer must be assessed.

References

1. Conway, M.: How Do Committees Invent? Datamation 14(4), 28–31 (1968)
2. Herbsleb, J.D., Grinter, R.E.: Architectures, Coordination, and Distance: Conway's Law and Beyond. IEEE Software 16(5), 63–70 (1999)
3. Clerc, V., Lago, P., van Vliet, H.: Global Software Development: Are Architectural Rules the Answer? In: Proc. of the 2nd International Conference on Global Software Engineering, pp. 225–234. IEEE Computer Society Press, Los Alamitos (2007)
4. Clerc, V., Lago, P., van Vliet, H.: Assessing a Multi-Site Development Organization for Architectural Compliance. In: Proc. of 6th Working IEEE/IFIP Conference on Software Architecture (WICSA 2007), p. 10. IEEE Computer Society Press, Los Alamitos (2007)

5. Herbsleb, J.D., Grinter, R.E.: Splitting the Organisation and Integrating the Code: Conway's Law Revisited. In: Proc. of the 21st International Conference on Software Engineering, pp. 85–95. IEEE Computer Society Press, Los Alamitos (1999)
6. Clements, P., Kazman, R., Klein, M.: Evaluating Software Architectures Methods and Case Studies. Addison-Wesley, Reading (2002)
7. Choi, H., Yeom, K.: An Approach to Software Architecture Evaluation with the 4+1 View Model of Architecture. In: Proc. of the 9th Asian-Pacific Software Engineering Conference, pp. 286–293. IEEE Computer Society Press, Los Alamitos (2002)
8. Dobrica, L., Niemelä, E.: A survey on software architecture analysis methods. IEEE Transactions on Software Engineering 28(7), 638–653 (2002)
9. Maranzano, J.F., Rozsypal, S.A., Zimmerman, G.H., Warnken, G.W., Wirth, P.E., Weiss, D.M.: Architecture Reviews: Practice and Experience. IEEE Software 22(2), 34–43 (2005)
10. Bass, L., Nord, R., Wood, W., Zubrow, D.: Risk Themes Discovered Trough Architecture Evaluations. In: Proc. of 6th Working IEEE/IFIP Conference on Software Architecture (WICSA 2007). IEEE Computer Society Press, Los Alamitos (2007)
11. Kazman, R., Bass, L.: Making Architecture Reviews Work in the Real World. IEEE Software 19(1), 67–73 (2002)
12. Herbsleb, J.D.: Global Software Engineering: The Future of Socio-technical Coordination. In: Proc. of Future of Software Engineering, pp. 188–198. IEEE Computer Society Press, Los Alamitos (2007)
13. Sangwan, R., Bass, M., Mullik, N., Paulish, D.J., Kazmeier, J.: Global Software Development Handbook. Auerbach Publications (2006)
14. Clerc, V.: Towards Architectural Knowledge Management Practices for Global Software Development. In: Proc. of the 8th International Workshop on Sharing and Reusing Architectural Knowledge, pp. 23–28. ACM, New York (2008)
15. Carl, D., Gupta, V., Javidan, M.: Power Distance. In: House, R.J., Hanges, P.J., Javidan, M., Dorfman, P.W., Gupta, V. (eds.) Culture, Leadership, and Organizations. The GLOBE Study of 62 Societies. Sage, Thousand Oaks (2004)
16. Cusumano, M.A.: Managing Software Development in Globally Distributed Teams. Communications of the ACM 51(2), 15–17 (2008)
17. Salger, F., Bennicke, M., Engels, G., Lewerentz, C.: Comprehensive Architecture Evaluation and Management in Large Software-Systems. In: Becker, S., Plasil, F., Reussner, R. (eds.) QoSA 2008. LNCS, vol. 5281, pp. 205–219. Springer, Heidelberg (2008)
18. Salger, F., Sauer, S., Engels, G.: An Integrated Quality Assurance Framework for Specifying Business Information Systems. In: Proc. of the Forum at the 21st International Conference on Advanced Information Systems, pp. 25–30. CEUR (2009)
19. Engels, G., Hess, A., Humm, B., Juwig, O., Lohman, M., Richter, J.P., Voß, M., Willkomm, J.: A Method for Engineering a True Service-Oriented Architecture. In: Proc. of the 10h International Conference on Enterprise Information Systems, vol. 2, pp. 272–281 (2008)
20. Haft, M., Humm, B., Siedersleben, J.: The Architect's Dilemma – Will Reference Architectures Help? In: Reussner, R., Mayer, J., Stafford, J.A., Overhage, S., Becker, S., Schroeder, P.J. (eds.) QoSA 2005 and SOQUA 2005. LNCS, vol. 3712, pp. 106–122. Springer, Heidelberg (2005)
21. Carmel, E.: Global Software Teams. Prentice Hall PTR, Englewood Cliffs (1999)
22. Sangwan, R., Ros, J.: Architecture Leadership and Management in Globally Distributed Software Development. In: Proc. of the 1st International Workshop on Leadership and Management in Software Architecture, pp. 17–22. ACM, New York (2008)