Paper presented at the EuroSPI 2010 conference, Grenoble, France, Sept. 1-3, 2010.

Contextualizing Agile Software Development

Philippe Kruchten, Ph.D., P.Eng.
Kruchten Engineering Services Ltd
Vancouver, BC, Canada

Abstract

This paper presents a contextual model for software-intensive systems development to guide the adoption and adaptation of agile software development practices. This model was found especially useful when the project context departs significantly from the "agile sweet spot", i.e., the ideal conditions in which agile software development practices originated from, and where they are most likely to succeed, "out of the box". This is the case for large systems, distributed development environment, safety-critical systems, system requiring a novel architecture, or systems with an unorthodox business model or governance model.

Keywords

Agile software development, software development process, process adaptation

1 Introduction

1.1 Agility defined

Drawing from earlier definitions from Jim Highsmith [1] or from Steve Adolph and the OODA loop [2], we define *agility* as "the ability of an organization to react to changes in its environment faster than the rate of these changes." This definition uses the ultimate purpose or function of being agile for a business, rather than defining agility by a labeled set of practices (e.g., you're agile when you do XP [3], Lean [4-6], or Scrum [7-9]) or by a set of properties defined in opposition to another set -- the agile manifesto approach [10]. This definition is not too far from that Kieran Conboy arrived at in his survey of agile process development literature [11].

An analogy could be the definition of a road. Would you define a road as something made of crushed rocks and tar, or define it as a surface that is black rather than white, flat rather than undulated, and with painted lines rather than monochrome? Or would you rather define a road as a component of a transportation system, allowing people and goods to be moved on the ground surface from point A to point B? And then let the properties or components of the road be derived from this, allowing some novel approaches in road design.

Note also that it is quite possible to adopt a labeled set of agile practices: Crystal [12], SCRUM, FDD [13], DSDM [14, 15], or a set of practices that perfectly conform to the agile manifesto [10] and not become agile. You would then "do agile", but you are not agile.

1.2 Agile methods successes

Agile software development methods do undoubtedly succeed in contexts that are identical or very similar to the contexts in which they have been initially created. As these contexts—the "agile sweet spot" [16]—are very frequent in software development, representing more than 70% of all software being developed, this may have led sometimes their proponents to a certain complacency: thinking that their method has universal value, that its represents some ultimate recipe, the holy grail of software engineering.

Agile methods may fail in various ways when they are applied "out of the box", i.e., with no or little adaptation, in contexts that are very far, at least on some dimension, from the context in which they have been originally created. Rather than an analysis of the root cause, this usually triggers screams of "you must have not done it right" by its proponents. And this again leads to discussions of "purity", "scrumbuts", etc. [39].

Agile methods can be stretched with variable success outside of the context in which they have been created; for example, scaling them up to larger projects, or across distributed teams. In our experience, the contextual factors that have the greatest risks of derailing agile projects are:

- size
- large systems with a lack of architectural focus
- software development not driven by customer demand
- lack of support from surrounding stakeholders, traditional governance
- novice team
- very high constraint on some quality attribute (safety-critical system, real-time constraints).

As noted by many authors in the last few years [17, 18], we cannot just rely on acts of faith by eloquent process gurus to help us define the adequate process, or set of practices outside of the agile sweet spot. Cold-headed, impartial investigation is required. Such research is generally not very easy to conduct; it is often qualitative, rather than quantitative, it draws more from social sciences than computer science, not easy to publish, not easy to carve down to masters' thesis bite size.

1.3 Overview of this paper

In this paper, drawing from our experience at Rational Software, and subsequently at KESL in consulting engagements with several organizations that have attempted to transition to agile software development methods, we present a contextual framework or model for situating agile practices. We define several factors characterizing software development context, which affect significantly the adoption of agile methods, especially when projects are outside of the "agile sweet spot" in which these methods have been defined and in which they operate at best. We describe four organizations that are developing software outside of this "sweet spot" and have run into difficulties applying agile software development methods "out of the box". We compare our model with other similar proposals. The ultimate purpose of this work, however, is to provide means for organizations to rapidly configure their method, by providing guidance on what agile practice to use in which circumstances.

2 Context Defined

Real estate agents in North America will tell you that only 3 factors do matter in their business: "Location, location, and location." For software process, we claim that only three factors matter: "context, context, and context." In *Voyage in the Agile Memeplex* [19], we had stressed the necessity to put our processes in context. but did not defined what "context" meant. It is very unfortunate that too many advocates of agile development practice are preaching good practices, but completely removed from the context in which they were proven to be successful. It turns some of their followers, several levels of transmission down to become just blind bigots, sometimes rabid bigots (see [19] for this phenomenon and the down side of *decontextualization*).

2.1 Two levels of context

There are 2 sets of factors that make up the context, which can be partitioned roughly in 2 sets: factors that apply at the level of whole organization/company, and factors that apply at the level of the project. In small organizations, with few software development projects, this distinction does not apply, and all factors are on the same level.

The organization-level factors (environment conditions) do influence heavily the project-level factors, which in turn should drive the process and practices that should be used

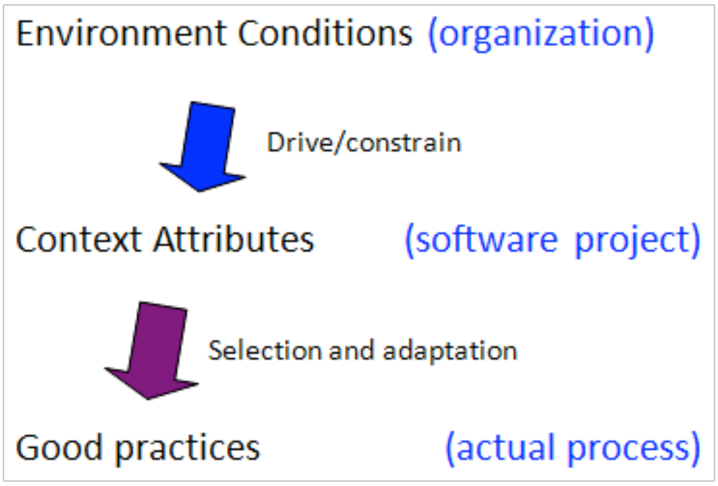

Fig. 1 – Environmental and project level context attributes

2.2 Organizational level: environmental factors

A certain number of factors are attached to the organization developing software, more than to the particular software development project.

1. Business domain

For what domain of activity is this organization developing software? Web-based systems, aerospace embedded systems, small hand-held instrumentation? Is software development the primary business activity of the organization, or at the other end, are we dealing with the IT organization of a business for which software is not at all the primary output. This factor more than any of the other 4 will condition, or constrain many of the project-level factors. For example, aerospace or biomedical instrumentation projects tend to be safety-critical.

2. Number of instances

How many instances of the software system (large or small) will be actually deployed? Are you building one single system, a dozen, a thousand, or millions? One-off systems are often internal to an organization, or developed on demand by a system integrator.

3. Maturity of organization

How long has that organization been developing software? How mature are the processes (and the people) relative to software development? Is the organization a small start-up, an SME (small or mediuym enterprise) or a large multinational firm? A small start-up is more likely to focus on a single, commercial piece of software, or more rarely open-source.

4. Level of innovation

How innovative is the organization? Are you creators or early adopters of new ideas and technologies? Or treading on very traditional grounds? Large IT organizations or government agencies tend to be rather risk-adverse and rarely tread on unchartered territory.

5. Culture

In which culture are the projects immersed? We are speaking here of both national culture and corporate culture? What are the systems of values, beliefs and behaviours that will impact, support or interplay with the software development practices? [20-24]

2.3 Project-level context attributes: the octopus model

At the level of a given software development project, we've identify empirically eight key factors:

1. Size

The overall size of the system under development is by far the greatest factor, as it will drive in turn the size of the team, the number of teams, the needs for communication and coordination between teams, the impact of changes, etc. [25, 26]. Number of person-months, or size of the code,, or development budget are all possible proxies for the size.

2. Stable architecture

Is there an implicit, obvious, de facto architecture already in place at the start of the project? Most projects are not novel enough to require a lot of architectural effort. They follow commonly accepted patterns in their respective domain. Many of the key architectural decisions are done in the first few days, by choice of middleware, operating system, programming languages, etc. [27] Some proponents of agile methods dismiss architecture at Big Up-Front Design, or YAGNI (You Ain't Gonna Need iI) [17, 28, 29] or confine it to a simple explanatory metaphor [3] (which is good, but not always sufficient).

3. Business model

What is the money flow? Are you developing an internal system, a commercial product, a bespoke system on contract for a customer, a component of a large system involving many different parties? Is it free, libre and open-source software (FLOSS)?

4. Team distribution

Linked often to the size of the project, how many teams are involved and are not collocated? This increases the need for more explicit communication and coordination of decisions, as well as more stable interfaces between teams, and between the software components that they are responsible for [30, 31]. Open-source development very often deals with scattered individuals, not teams.

5. Rate of change

Though agile methods are "embracing changes", not all domains and system experience a very rapid pace of change in their environment. How stable is your business environment and how much risks (and unknowns) are you facing? There are still projects with very stable requirement definitions.

6. Age of system

Are we looking at the evolution of a large legacy system, bringing in turn many hidden assumptions regarding the architecture, or the creation of a new system with fewer constraints?

7. Criticality

How many people die or are hurt if the system fails? Documentation needs increase dramatically to satisfy external agencies who will want to make sure the safety of the public is assured [32-35].

8. Governance

How are projects started, terminated? Who decides what happens when things go wrong? How is success or failure defined? Who manage the software project managers? [36-38]

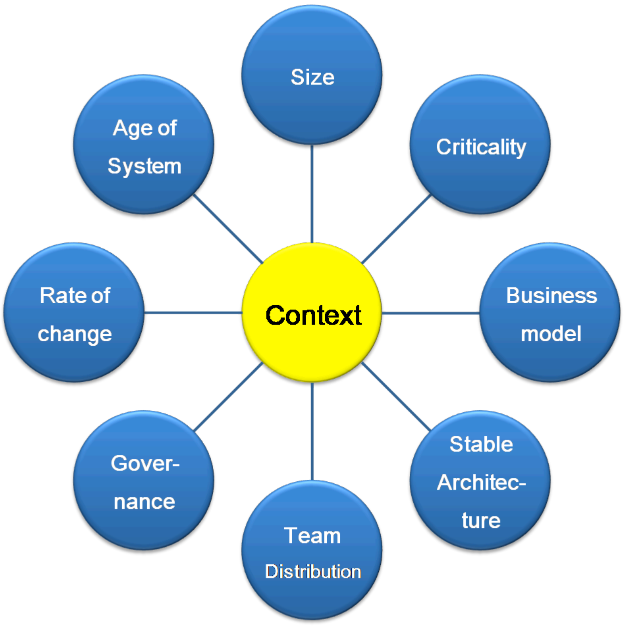

Fig.2 – The octopus: 8 key contextual factors for agile software development

2.4 Relationship between the two sets of factors

As you can expect, the first set of factors (organizational level) impacts and constrains the second set. But there is still a wide range of variation in this second set inside any given organization, especially large software development shops (a.k.a. system integrators) offering "bespoke" software development services.

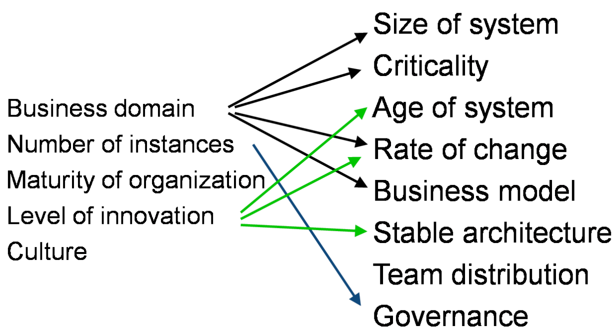

Fig.3 - Relationships between the 2 sets

If your business domain is "aerospace-onboard software", this will pre-condition several project-context factors: criticality, size, business model, etc. This in turn will make some practices suitable or not, will influence amount and type of documentation, the "level of ceremony" required by a given project.

2.5 The agile sweet spot

Figure 4 shows, based on this model, the "agile sweet spot," i.e., the conditions under which most "labelled" agile software development have been developed, and for which success is pretty much assured [16]. This would be the case for example of a web-based e-commerce site, built on dot-net technology by a small team, interacting with their customers.

System Size	• 0 ..12 … 300
Criticality	• Simple, $ losses, … deaths
System Age	• Exploratory, greenfield, legacy maintenance
Rate of change	• Low, medium, high
Business model	• In house, Open source, ….
Stable architecture	• Stable, changed, new
Team distribution	• Collocated, …, …, offshore outsource
Governance	• Simple rules, …, SOX, …

Fig. 4 – A particular context: the agile sweet spot (in blue)

The “agile sweet spot” tends to be for collocated team, of less than 15 people, doing greenfield development for non safety-critical system, in rather volatile environment; the system architecture is defined and stable, and the governance rules straightforward. Again, we are not trying to say that agile practices do not work outside of this sweet spot, but that many are challenged, will need some adaptation, and in some cases not be suitable.

3 *Four Agile projects outside of the sweet spot*

In our consulting practice we have come across 4 organizations in the last 10 years that have tried to embrace agile software development methods, but have run into difficulties [39]. We found in many cases that this was mostly because they were in some way outside of the “agile sweet spot”.

1. Project FIN

This large legacy software was being re-implemented in Java: 50 developers, collocated for about 18 months. Although progressing very well for the first 6 months using a combination of XP and Scrum, this project “hit a wall” after a while, mostly because of its failure to put in place a solid underlying architecture, driven by a naïve belief that a solid software architecture would gradually emerge as the result of bi-weekly refactorings.

2. Project FAC

A large factory automation project, with again a large amount of legacy software, it felt outside of the sweet spot on several other dimensions: safety-critical, very low rate of change, and no concept of “customer” to interact with, since many aspects are just related to physics. Software cannot be tested in real environment, and there is only one release opportunity per year.

3. Project FLY

Multiple inter-related legacy projects and “greenfield” projects, some deployed in the cockpit of aircraft, having to comply to the highest safety critical standards [32], and therefore requiring large amount of very detailed documentation, in particular traceability of artifacts, which run counter to the agile manifesto [10] and created cultural clashes in the development teams.

4. Project ANA

Small start-up company developing novel algorithms to support security trading, but not driven by customers’ demand, but driving the development internally based on models from physics, hence putting many of the agile practices at odds.

In each of these four projects, the way forward was defined the following pattern:

A) develop a clearer understanding of the contextual factors, using our model for example, allowed the project(s) to focus on what was specific to their project, and

b) specify the areas needing improvement from a software process perspective,

c) and finally select carefully the agile practices that would solve the issues they actually were facing, rather than blind application of a labeled agile methods in its totality, all practices lumped together; for example, all XP practices.

4 Review of other similar contextual models

Other authors have attempted to define the *context* of a software process, to define the main factors that affect the process they use/could use/would use. Let us examine a few such models in the agile adoption arena:

1. Boehm-Turner

In their book *Balancing agility and discipline* [39], Barry Boehm and Richard Turner used 5 factors to contrast software process/methods between what they call “plan-driven methods” and agile methods:

1. Size,
2. Criticality,
3. Personnel (their skill, know-how),
4. Dynamism (rate of change) *and*
5. Culture (of the team: thriving on chaos or on order).

Their 5 factors provided us with starting point, but we found that: a) they were mixing organization-level issues, with project-level issues, and b) they were lumping together too many aspects into the factors Personnel and Culture.

2. Cockburn & Crystal

In the Crystal family of processes [12], Alistair Cockburn defines different processes based on

1. Size,
2. Criticality, *and*
3. Skills.

The first two are fully aligned with ours, but we found the ’Skills’, which matches ‘Personnel’ in Boehm-Turner, difficult to use in practice. It is not quite a linear scale. We do capture some of it under “Maturity of the organization” however.

3. Ambler (IBM) and “Agile@Scale”

Closer to our views are those of Scott Ambler in *Agility at Scale* [40] which is summarized by his table reproduced in fig.5. We seem to cover mostly the main grounds, though with some small differences here and there, as shown in table 1. Scott Ambler seems to focus on scaling agility to larger projects, whereas I am mostly attempting to adjust to the context, regardless of the size or ambition.

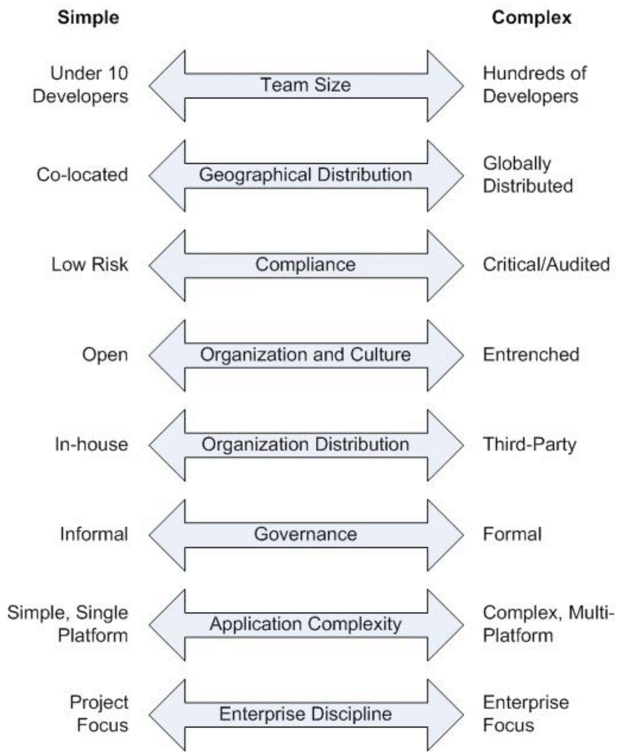

Fig.5- Scott Ambler's scaling factors (from fig. 4 in [40])

Table 1 – Comparing Agile@Scale [40] and our contextual model

Ambler	*Kruchten*	*Comment*
Team Size	Size	Number of people or SLOCs or function points? Either one is a possible indicator of size, as they are strongly correlated.
Geographical Distribution	Team distribution	(Same concept)
Compliance	Criticality	Not quite identical, but linked: critical system have to be compliant to standards and regulations: SOX [42], Cobit, [43], Basel II [43], or Do178B [32] depending on the industry
Organization & Culture	*Culture*	Rarely specific to one project
Organization distribution	Business model	(Same concept)
Application complexity	Stable architecture, Size	Could be also linked to innovation level
Enterprise discipline	*Maturity of the organization*	May vary across projects
Governance	Governance	(Same concept)
	Age of system	Issues with large legacy system
	Rate of change	Drives iteration duration, and the nature of the feedback loop

5 Future work

The authors and several colleagues have used the model in an ad hoc fashion, mostly applying judgement calls, and past (negative) experience to decide which practice could be used in a given combination of project factors. The ideal situation would be to build a kind of recommending system: a tool to which we would provide values for the 5 + 8 factors, that would give an indication of which practices are usable, which are not, which would require adaptation or special consideration, as the starting point for a process configuration. The main hurdle is to objectively populate such a table with reliable data, supported by evidence, and not just guts' feelings, as the number of data points is rather large. A short list of agile practices would be about 40: short iterations, managing a backlog, daily stand-up meetings, test-driven development, continuous integration, etc. With only 3 or 4 values for each of the 8 factors in the "octopus" we have already a thousand data points. Some cases are easier than others: short iterations, or conducting retrospectives have rather wide applicability. Table 2 shows an embryo of what such a table could contain.

Table 2 – Guidance for process configuration

Factor	Size			Criticality			Distribution				Rate of change		Age of system		…
Values	S	M	L	L	M	H	-	M	L	XL	L	H	G	B	
Iterations															
Daily standup															
Retrospective															
Pair prog.															
Backlog															
Metaphor															
Monthly release to users															
More agile practices…															

Legend:

- Practice is not affected by the factor
- Practice needs caution or adaptation
- Practice not useful, or require considerable adaptation
- Practice to avoid, could be dangerous or counterproductive
- Not enough evidence to conclude

Even in this simple form, the table generates lots of discussions and controversies, and each non-green cell must have an explanation, or rationale, and some caveats. So it should not be used as some kind of prescription, but more as a way to focus the reflection and discussion on the value of practices. Fortunately, as agile methods have crossed the chasm and become mainstream, there is a growing body of evidence to draw from, in particular experience reports, such as the ones collected through agile conference [45, 46], even though many tend to be heavily biased towards the "green", and complemented by studies of researchers, looking at the impact of going outside the sweet spot [47], or just plain old critiques of agile methods [48]. It should be useful to also submit non-agile practices to the same filter.

6 Conclusion

In assisting software organization adopting agile software development processes, we found valuable to first define the context using our model with several factors, then understand in which dimension(s) the project felt outside of an ideal agile sweet spot, and then in turn drive the adoption and possible adaptation of agile practices to this context, rather than a forced-fit, en-bloc adoption of all practices falling under a certain agile label, in the sometimes naïve hope that it will cure all ills. But again, agility should not be defined in terms of practices, but as the ability of an organization to react to changes in its environment faster than the rate of these changes.

7 References

1. Highsmith, J.A.: Agile software development ecosystems. Addison-Wesley, Boston (2002)
2. Adolph, W.S.: What lessons can the agile community learn from a maverick fighter pilot? In: Melnik, G. (ed.): Agile 2006 conference. IEEE Comp Soc, Minneapolis, MN, USA (2006) 94-99
3. Beck, K.: Extreme Programming Explained: Embrace Change. Addison-Wesley, Boston (2000)
4. Poppendieck, M., Poppendieck, T.: Lean Software Development - An Agile Toolkit. Addison-Wesley, Boston, MA (2003)
5. Poppendieck, M., Poppendieck, T.: Implementing Lean Software Development--From Concept to Cash. Addison Wesley, Upper Saddle River, NJ (2009)
6. Poppendieck, M., Poppendieck, T.: Leading Lean Software Development: Results Are not the Point. Addison-Wesley Professional, Boston, MA (2009)
7. Cohn, M.: Succeeding with Agile-Software development using Scrum. Pearson Education, Boston, MA (2009)
8. Schwaber, K., Beedle, M.: Agile Software Development with SCRUM. Prentice-Hall, Upper Saddle River, NJ (2002)
9. Schwaber, K.: The Enterprise and Scrum. Microsoft Press, Redmond, WA (2007)
10. Agile Alliance: Manifesto for Agile Software Development (2001). At http://www.agilemanifesto.org/
11. Conboy, K.: Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development. Information Systems Research 20 (2009) 329-354
12. Cockburn, A.: Crystal Clear--A Human-Powered Methodology for Small Teams. Addison-Wesley, Boston, MA (2004)
13. Palmer, S.R., Felsing, J.M.: A Practical Guide to Feature-Driven Development. Prentice-Hall, Upper Saddle River, NJ (2002)
14. Stapleton, J.: DSDM, Dynamic Systems Development Method: The Method in Practice. Addison-Wesley, Reading, MA (1998)
15. Stapleton, J. (ed.): DSDM Business Focused Development. Addison-Wesley, London, UK (2003)
16. Kruchten, P.: Scaling down projects to meet the Agile sweet spot. The Rational Edge (August 2004) At: http://www-106.ibm.com/developerworks/rational/library/content/RationalEdge/aug04/5558.html
17. Abrahamsson, P., Ali Babar, M., Kruchten, P.: Agility and Architecture: Can they Coexist? IEEE Software 27 (2010) 16-22
18. Abrahamsson, P., Salo, O., Ronkainen, J., Warsta, J.: Agile Software Development Methods-Review and Analysis. VTT Technical Research Centre of Finland, Oulu, Finland (2002)
19. Kruchten, P.: Voyage in the Agile Memeplex: Agility, Agilese, Agilitis, Agilology. ACM Queue 5 (2007) 38-44 also at http://queue.acm.org/detail.cfm?id=1281893
20. Borchers, G.: The Software Engineering Impacts of Cultural Factors on Multi-cultural Software Development Teams. 25th International Conference on Software Engineering (ICSE'03). IEEE Computer Society, Portland, OR (2003) 540-545
21. Sharp, H., Woodman, M., Hovenden, F., Robinson, H.: The Role of 'Culture' in Successful Software Process Improvement. 25th Euromicro Conference (EUROMICRO '99), Vol. 2. IEEE-CS, Milan, Italy (1999) 2170-2176
22. Hsieh, Y., MacGregor, E.L., Kruchten, P.: Intercultural Factors in Global Software Development. In: Kasap, S. (ed.): 18th Annual Canadian Conference on Electrical and Computer Engineering (CCECE'05). IEEE, Saskatoon, SK (2007)
23. Kruchten, P.: Analyzing Intercultural Factors Affecting Global Software Development. In: Damian, D., Lanubile, F. (eds.): 3rd International Workshop on Global Software Development (GSD2004), Collocated with ICSE 2004, Edinburgh, Scotland, IEE (2004) 59-62
24. Sharp, H., Robinson, H., Woodman, M.: Software engineering: Community and culture. IEEE Software 17 (2000) 40-47

25. Larman, C., Vodde, B.: Scaling Lean & Agile Development--Thinking and Organizational Tools for Large-Scale Scrum. Addison-Wesley Professional, Boston (2008)

26. Leffingwell, D.: Scaling Software Agility: Best Practices for Large Enterprises. Addison-Wesley Professional, Boston (2007)

27. Kruchten, P.: The Rational Unified Process: An Introduction. Addison-Wesley Professional (2003)

28. Madison, J.: Agile-Architecture Interactions. IEEE Software 27 (2010) 41-47

29. Blair, S., Watt, R., Cull, T.: Responsibility-Driven Architecture. IEEE Software 27 (2010) 26-32

30. Ågerfalk, P.J., Fitzgerald, B.: Flexible and Distributed Software Processes: Old Petunias in New Bowls? Comm. ACM 49 (2006) 27-34

31. Lings, B., Lundell, B., Ågerfalk, P.J., Fitzgerald, B.: Ten strategies for successful distributed development. Proceedings of the IFIP WG 8.6 Conference, Galway, Ireland. Vol. 206. Springer-Verlag (2006) 94-112

32. RTCA/DO-178B Software Considerations in Airborne Systems and Equipment Certification. Radio Technical Commission for Aeronautics, Washington, DC (1992)

33. Leveson, N.G.: Safeware: System Safety and Computers. Addison-Wesley, Reading, MA (1995)

34. Paige, R.F., Chivers, H., McDermid, J.A., Stephenson, Z.R.: High-Integrity Extreme Programming. 2005 ACM Symposium on Applied Computing. Santa Fe, New Mexico, USA, ACM (2005) 1518-1523

35. Poppendieck, M., Morsicato, R.: Using XP for Safety-Critical Software. Cutter IT Journal 15 (2002) 12-16

36. Cockburn, A.: A Governance Model for Incremental, Concurrent, or Agile Projects. CrossTalk: The Journal Of Defense Software 19 (2006) 13-17

37. Dubinski, Y., Yaeli, A., Feldman, Y., Zarpas, E., Nechushtai, G.: Governance of Software Development: The Transition to Agile Scenario. In A. Cater-Steel (Ed.), IT Governance and Service Management Frameworks and Adaptations, Idea Group Publishing, Information Science Publishing, IRM Press, Hershey, PA (2009) 266-284

38. Dubinsky, Y., Kruchten, P.: Software Development Governance (SDG): Report on 2nd ICSE Workshop. ACM SIGSOFT Software Engineering Notes 24 (2009) 46-47

39. Hoda, R., Kruchten, P., Noble J., Marshall, S.: Agility in Context, in: Rinard, M. (Ed.): Proceedings of OOPSLA 2010 (part of SPLASH), Las Vegas, October 17-21, 2010, ACM (2010)

40. Boehm, B.W., Turner, R.: Balancing Agility and Discipline--A guide for the perplexed. Addison-Wesley, Boston, MA (2003)

41. Ambler, S.W.: Agility at Scale: Become as agile as you can be. IBM, Toronto (2009). eBook at http://www.internetevolution.com/ebook/ebookibm7/

42. US Public Law 107-- 204 - Sarbanes-Oxley Act of 2002. US Government Printing Office, Washington (2002).

43. CobiT 4.0--Control Objectives for Information and related Technology--Control Objectives, Management Guidelines, Maturity Models. IT Governance Institute. Rolling Meadows, Il., USA (2005)

44. Basel II--Revised international capital framework, Bank for International Settlements, Basel, CH (2004).

45. Melnyk, G, Poppendieck, M. (Eds.): Proceedings of Agile 2008 Conference, Toronto, Aug. 4-8, IEEE Computer Society, Los Alamitos (2008)

46. Dubinsky, Y., Dybå, T., Adolph, S., Sidiky, A. (Eds.): Proceedings of Agile 2009 Conference, Chicago, Aug 24-28, IEEE Computer Society, Los Alamitos (2009)

47. Šmite, D., Moe, N.B., Ågerfalk, P.J. (Eds.): Agility across Time and Space -- Implementing Agile Methods in Global Software Projects, Springer-Verlag, Berlin (2010)

48. Stephens, M., Rosenberg, D.: Extreme Programming Refactored--The Case Against XP. Apress LP, Berkeley, CA (2003)

8 Author CV

Philippe Kruchten

Philippe Kruchten is the founder and principal at KESL in Vancouver, Canada. He is also professor of software engineering at the University of British Columbia in Vancouver, which he joined in 2004 after a 32-year career in the software industry, developing systems in telecommunications, defense and aerospace. He holds an NSERC chair in design engineering. His main interests are in software architecture, software project management and software development processes. During his 17 years with Rational Software (now part of IBM) he led the development of the Rational Unified Process® (RUP®), an iterative and incremental software development process. He is the co-founder and secretary of the IFIP working group 2.10 on Software Architecture, and the co-founder and past chair of Agile Vancouver, a special interest group focusing on agile software development approaches.
Dr. Kruchten has a mechanical engineering degree from Ecole Centrale de Lyon (France), and a doctorate degree form Ecole Nationale Supérieure des Télécommunications in Paris. He is a licensed professional software engineer in British Columbia, Canada, an IEEE Certified Software Development Professional (CSDP) and even a Certified Scrum Practitioner (CSP). He is a member of IEEE, ACM, INCOSE and AIS.