Method and Developer Characteristics for Effective Agile Method Tailoring: A Study of XP Expert Opinion

KIERAN CONBOY
National University of Ireland, Galway
and
BRIAN FITZGERALD
University of Limerick

It has long been acknowledged that software methods should be tailored if they are to achieve optimum effect. However comparatively little research has been carried out to date on this topic in general, and more notably, on agile methods in particular. This dearth of evidence in the case of agile methods is especially significant in that it is reasonable to expect that such methods would particularly lend themselves to tailoring. In this research, we present a framework based on interviews with 20 senior software development researchers and a review of the extant literature. The framework is comprised of two sets of factors—characteristics of the method, and developer practices—that can improve method tailoring effectiveness. Drawing on the framework, we then interviewed 16 expert XP practitioners to examine the current state and effectiveness of XP tailoring efforts, and to shed light on issues the framework identified as being important. The article concludes with a set of recommendations for research and practice that would advance our understanding of the method tailoring area.

Authors' addresses: K. Conboy, Business Information Systems Group, Department of Accountancy and Finance, National University of Ireland, Galway, University Road, Galway, Ireland; email: Kieran.Conboy@nuigalway.ie; B, Fitzgerald, Lero-Irish Software Engineering Research Centre, University of Limerick, Limerick, Ireland; email: bf@ul.ie.

1. INTRODUCTION

Extreme Programming (XP), along with a number of other agile methods, has emerged in recent years as a popular approach to software development. Proponents of the method claim it solves many of the problems endemic to the field for over 40 years—namely, that systems cost too much, take too long to develop, and do not serve their intended purpose when eventually delivered. The aim of this article is to get a better understanding of XP[1] tailoring in practice and how it can be improved in the future. Also, there has been very little research to date on method tailoring, and we sought to address this by choosing to focus on tailoring from two perspectives: first, characteristics of the method itself; and second, characteristics of the actual developers involved in tailoring. The specific objectives of the article are to

(1) assess how amenable XP is to tailoring, and to develop a set of recommendations for its improvement in this regard, and
(2) investigate how developers are undertaking XP tailoring efforts and to develop a set of best practices for developers to follow.

For our theoretical base we proposed a conceptual framework drawn from existing method tailoring literature, and conducted interviews with 20 expert researchers to further validate the framework. Using the framework as an analytical lens, we then interviewed 16 experienced XP practitioners to assess the current state of XP tailoring. The article concludes with a set of recommendations to address shortcomings in XP in this area, and a set of developer best practices for XP tailoring.

1.1 Motivation for This Research

There has been a long-standing acknowledgment that software methods need to be tailored for use, the essence of which is captured well by De Marco [1982], p. 13:

> I find myself more and more exasperated with the great inflexible sets of rules that many companies pour into concrete and sanctify as methodologies. Use the prevailing methodology only as a starting point for tailoring.

This continues as a persistent theme in software engineering research, as evidenced by the contention by Sommerville and Ransom [2005], p. 93:

> It is a truism that any method has to be adapted for the particular circumstances of use.

Despite these contentions, little research has been carried out into method tailoring to date, especially in the context of agile methods [Aydin et al. 2004]. Although some research has described tailoring efforts, these were usually limited to single case studies. Also, such research has tended to treat method tailoring as a homogeneous concept. Our primary motivation in this study was

---

[1]XP has provided the focus for over 20 texts, an annual conference, and indeed the vast majority of agile method academic research and practice to date. Given its popularity in both research and practice, we chose to focus on XP in this study.

to investigate the method tailoring topic in more depth. Thus, we considered the tailoring issue from two complementary perspectives—that of the characteristics of the actual method being tailored, and also the characteristics of individual developers responsible for method tailoring. This rich two-faceted research approach has not been a feature of previous research on the method tailoring topic.

There is a common misconception that agile methods are centered around improvisation and care-free deviation from rules and regulations. However, Beck has dismissed this, stating that agile "is not an excuse for unilateral behaviour" and he described agility versus discipline as a "false dichotomy." In fact, he has argued that agility "is only possible through greater discipline on the part of everyone involved" [Beck and Boehm 2003]. The need for discipline has been stressed by many key texts across a broad range of agile methods (e.g., Schwaber [1996]; Cockburn [2001, 2002]; Schwaber and Beedle [2002]; Beck and Andres [2004]). Given that tailoring of a method is a key part of the method implementation process, it is important to understand whether tailoring of agile methods is conducted in a disciplined and structured manner. In the literature that does exist on this topic, tailoring of XP and other agile methods seems to be quite a contentious topic. Some believe that flexibility is one of the key selling points of agile methods (e.g., Beck [2000]; Schwaber and Beedle [2002]), while others have argued that these methods are not actually flexible (e.g., Stephens and Rosenberg [2003]; Henderson-Sellers and Serour [2005]). Documenting the opinions of experts on XP tailoring, as done in this article, should contribute to advancing current thinking on this debate.

Advocates of XP have suggested it can solve the multitude of problems affecting software development, namely, time and budget overruns, inferior and ineffective software, and dissatisfied developers, customers, and users (e.g., Beck [2000]). However, method tailoring theory suggests that, no matter how well crafted, there is no single method that provides an exact fit for the needs of every project Iivari [1989]; Brinkkemper [1996]. Therefore, if XP is to be regarded as a truly mainstream method, it should be highly amenable to tailoring. This study seeks to identify if this is indeed the case, and if not, what can be done to improve XP in this regard.

Another key motivation of the study is that, where tailoring is concerned, agile methods introduce many new problems and complexities, or at least exacerbate existing ones. Because agile practices "value people over processes and tools" [Fowler and Highsmith 2001] and "turn up the dial" on social, tacit interaction, tailoring of those practices needs to be much more sensitive to personal characteristics and team dynamics. Unfortunately these traits are often highly subtle, intangible, and difficult to identify. In addition, agile methods tend to increase the involvement of other stakeholders in the development process. For example, customers and users play a much more significant, integrated, and continuous role when XP is adopted. Tailoring in this case needs to consider not simply the developers but also the needs and characteristics of these other stakeholders and the underlying complexities within their organizations.

There is a further theoretical motivation behind this study. Agile methods are labeled as *agile* because of their ability to handle changing requirements

quickly and effectively [Schwaber 1996; Cockburn 2001, 2002; Schwaber and Beedle 2002; Beck and Andres 2004]. However, it is logical to expect that anything labeled as *agile* should itself be flexible and amenable to tailoring. That is, a method that purports to be agile should possess both abilities: first, it should allow changing requirements, and second, it should itself be flexible in that it can be continuously changed and customised. Previous studies (e.g., Henderson-Sellers and Serour [2005]; Conboy [2006]) have argued that existing research on agile methods has largely focused on the first but not the second ability. This study aims to increase our understanding of the second, analyzing XP's amenability to tailoring.

A further practical motivation for this study originated from a survey of Irish organizations involved in agile software development, where tailoring of agile methods was rated as the primary concern, thus confirming the importance of this topic in practice also. When asked why tailoring was such an issue, organizations referred to a lack of knowledge about how tailoring should be done, and in some cases to previous tailoring efforts that had failed. In particular, many organizations had invested considerable resources in agile method adoption and training, but postimplementation use of the respective methods became so sporadic and disjointed that the transition to agile was abandoned. These companies were eager to get information as to how agile method experts are tailoring these methods and to obtain a a set of best practices to assist in tailoring efforts. We seek to provide this information in this study.

1.2 Extreme Programming

Since the early years of computing, software development projects generally have tended to be troubled by time and budget overruns, inferior and ineffective software, and dissatisfied developers, customers, and users [Brooks 1975; Lehman 1978; Glass 1991; Johnson 1995; Linberg 1999; Keil et al. 2000; Robey and Keil 2001]. Many methods, method hybrids, and method variants have been developed and implemented in the hope of overcoming these problems [Jenkins et al. 1984; Necco et al. 1987; Hardy et al. 1995] in the hope of finding what Brooks [1987] famously termed *the silver bullet*. The late 1990s and early 2000s have seen the emergence of agile methods, which seek to "restore credibility to the word *method*," and to eradicate the problems that have hindered software development for so long [Fowler and Highsmith 2001]. A number of methods are included in the agile family, the most notable being *Extreme Programming (XP)* [Beck 2000], *Scrum* [Schwaber and Beedle 2002], the *Dynamic Systems Development Method* (DSDM) [Stapleton 1997], *Crystal* [Cockburn 2001], *Agile Modeling* [Ambler 2002], *Agile Project Management* (APM) [Highsmith 2004], *Feature-Driven Design* [Coad and Palmer 2002], and *Lean Software Development* (LSD) [Poppendieck 2001].[2] These methods represent quite a popular initiative that complements previous critiques of formalized methods (e.g., Baskerville et al. [1992]; Fitzgerald [1996]; Truex et al. [1999]), and

---

[2]Methods are often distributed and communicated in different ways, namely, through manuals, research articles, consulting, mentoring, etc. In the interests of consistency, this study refers to the version of each method as documented in the associated references above.

> **The planning game:** a quick determination of the scope o f the next software release, based on a combination of business priorities and technical estimates. It is accepted that this plan will probably change.
> **Small releases:** put a simple system into production quickly, then release new versions on a very short cycle.
> **Metaphor:** guide all development with a simple shared story of how the whole system works.
> **Simple design:** the system should be designed as simply as possible at any given moment in time.
> **Testing:** programmers continually write tests, which must be run flawlessly for development to proceed. Customers write function tests to demonstrate the features implemented.
> **Refactoring:** programmers restructure the system, without removing functionality, to improve nonfunctional aspects (e.g., duplication of code, simplicity, flexibility).
> **Pair-programming:** all production code is written by two programmers at one machine.
> **Collective ownership:** anyone can change any code anywhere in the system at any time.
> **Continuous integration:** integrate and build the system every time a task is completed—this may be many times per day.
> **40-Hour week:** work no more than 40 hours per week as a rule.
> **On-site customers:** include an actual user on the team, available full time to answer questions.
> **Coding standards:** adhere to coding rules that emphasize communication via program code.

Fig. 1.   Key practices of XP (adapted from Beck [2000]).

have been well received by practitioners. There is also evidence to suggest that use of agile methods has been growing rapidly since their inception [Schwaber and Fichera 2005; Ambler 2007; Tan and Teo 2007; Vijayasarathy and Turk 2008].

XP originated from an internal payroll system project at Chrysler in 1996–1997. The project initially suffered from many of the symptoms associated with traditional software projects, and so the developers involved, including Kent Beck, constructed a new "common-sense" approach [Beck 2000]. This was comprised of five key values, namely, communication, feedback, simplicity, courage, and respect. These in turn were enacted by 12 key practices, summarized in Figure 1. These collectively became known as *XP*, which is comprehensively described by Beck [2000, p. 12], as "a light-weight methodology for small-to-medium-sized teams developing software in the face of vague or rapidly-changing requirements." Beck [2000] explicitly acknowledged that XP is not a set of revolutionary new development techniques. Rather, it is a set of tried and trusted principles, well established as part of the conventional wisdom of software engineering, but taken to an extreme level—hence the name *extreme programming*.

An extended version of XP was introduced in 2004, containing 24 practices [Beck and Andres 2004]. At the time of that study, however, it proved difficult to find any project teams using the new version of the method. Furthermore, while many of the academic interviewees had extensive theoretical and applied knowledge of the original version, few had any substantive exposure to the 2004 version. In addition, very little research has focused on the new version. All of these issues could simply be due to the fact that any method takes time to gain traction and popularity, and it may only be a matter of time before use of the revised version reaches the same levels as use of the original. This being the case, we decided to study tailoring of the original set of practices, given that there is already a well-established audience for the work and an existing population of method experts to choose from. Also, given that the study addresses long-term implementation and tailoring of a method, even if one were to find teams using the new practices, it is unlikely that they would have been using them for a period of time sufficient to critically reflect on such long-term issues.

1.3 Method Tailoring Theory

To overcome the many problems traditionally associated with software development, there is a tendency to replace older methods with new and apparently improved alternatives. However, constantly striving to derive the ultimate method or "silver bullet" is regarded by some as somewhat misguided (e.g., Iivari [1989]; Hardy et al. [1995]; Russo et al. [1995]; Brinkkemper [1996]). While method adherence may yield many benefits, the notion that any one method is universally superior or even universally applicable has been viewed as fallacious. It is widely accepted that almost all software development projects are unique, and that the choice of method or method variant is dependent on many organizational, technical, or human factors, and the nature of the system being developed. As a result, it is rare that a method is most effectively deployed in its original, textbook format. Instead the optimum solution is a tailored or engineered approach to suit each project context [Iivari 1989; Hardy et al. 1995; Russo et al. 1995; Brinkkemper 1996]. Therefore, a key attribute of an effective method is that it can be tailored effectively.

As with many themes and issues in the software development literature, tailoring of software engineering methods has been referred to by many different terms, including *context-based method use* [Rolland and Prakash 1996], *method adaptation* [Baskerville and Stage 2001], *method assembly* [Brinkkemper et al. 1998], *method configuration* [Karlsson and Agerfalk 2004], and *scenario use* [Offenbeek and van Koopman 1996]. However, these can be classified into one of two overarching approaches—namely, contingency-based method selection (e.g., Naumann et al. [1980]; Davis [1982]; Gremillion and Pyburn [1983]; Sullivan [1985]; Benyon and Skidmore [1987]; Iivari [1989]) and method engineering (e.g., Kumar and Welke [1992]; Tolvanen and Lyytinen [1993]; Harmsen et al. [1994]; Brinkkemper [1996]). Contingency-based method selection is based on the premise that, rather than accepting a software development method as being universally applicable, the team

should choose a method from a broad portfolio of development methods to suit each different project context. Method engineering on the other hand, is a metamethod process, where instead of selecting a method from an available library, a new one is constructed or "engineered" from the ground up using existing "method fragments"[3] (Brinkkemper 1996).

Existing research suggests that both contingent-based method selection and method engineering are usually conducted in an ad hoc, unstructured format, and that developers learn little about each tailoring effort as they progress from one project to the next [Fitzgerald et al. 2002; Mirbel and Ralyte 2006]. The literature contains little insight into why this is the case, and few methods contain any characteristics which aid tailoring [Iivari 1989]. Furthermore, research suggests that developers need to improve they way in which they tailor these methods [Kumar and Welke 1992].

2. RESEARCH APPROACH

To achieve the objectives discussed earlier, this study was conducted in two phases (illustrated graphically in Figure 2). In the first we identified a set of method and developer characteristics which contribute to effective method tailoring, and derived a consolidated conceptual framework. This was achieved through an iterative process, combining (i) a comprehensive review and synthesis of the extant method tailoring literature, and (ii) interviews and other more informal communication exchanges with 20 expert researchers. The researchers provided continual input into the development of the framework, and opinions on the current literature and its applicability or relevance. Based on the formal interviews, and subsequent followup communications, the framework was continually refined and extended. The decision to involve researchers at this stage was based on a number of factors. Perhaps of most significance was the fact that much of the extant method tailoring research has been conducted on teams using large, formal, heavyweight approaches. Developing a framework, based on this body of knowledge, but for evaluating more lightweight, agile approaches required careful consideration. Given the absence of substantial conceptual research on agile method tailoring, the use of researchers to advise on changes made to accommodate agile methods, and to ultimately validate the framework, was considered very beneficial.

Using the proposed framework, the second phase of the research sought to develop an understanding of the current state of XP tailoring. This was achieved by interviewing 16 expert XP practitioners. The framework components were drawn on to formulate interview questions. As with any study, the framework in this study provided a set of "intellectual bins" [Miles and Huberman 1999] to structure the collection and analysis of practitioner data.

---

[3]A method fragment is defined as any "subcomponent" of a method [Brinkkemper 1996]. Logically, the definition of *method fragment* depends on one's interpretation of *method*, itself a term which has been interpreted in many different ways [Connors 1992; Wynekoop and Russo 1995; Brinkkemper 1996; Fitzgerald et al. 2002; Avison and Fitzgerald 2003]. For the purposes of this study, this subcomponent can be an artifact, action, goal, or value within a method [Ågerfalk and Wistrand 2003; Ågerfalk and Fitzgerald 2005].

PHASE 1

Method Tailoring Literature

Conceptual Framework

Derivation of Questions for Phase 2 Practitioner Interviews

Researcher	Institution
Ivan Aaen	Aalborg University
Pekka Abrahamsson	VTT Finland
Richard Baskerville	Georgia State University
Michael Cusumano	Sloan MIT
Rob Fichman	Boston College
Guy Fitzgerald	Brunel University
Jim Herbsleb	Carnegie Mellon University
Mike Holocombe	University of Sheffield
Linda Levine	Software Engineering Inst.
Kalle Lyytinen	Case Western Reserve
Lars Mathiassen	Georgia State University
Peter Middleton	Queens University Belfast
Mark Paulk	Carnegie Mellon University
Jan Pries-Heje	IT University of Copenhagen
Nancy Russo	Northern Illinois University
Helen Sharp	Open University
Ian Sommerville	Lancaster University
Duane Truex	Georgia State University
Laurie Williams	North Carolina State
Robert Zmud	University of Oklahoma

PHASE 2

Evaluation of XP Tailoring

Recommendations for XP Researchers to Improve Tailoring

Recommendations for XP Practitioners to Improve Tailoring

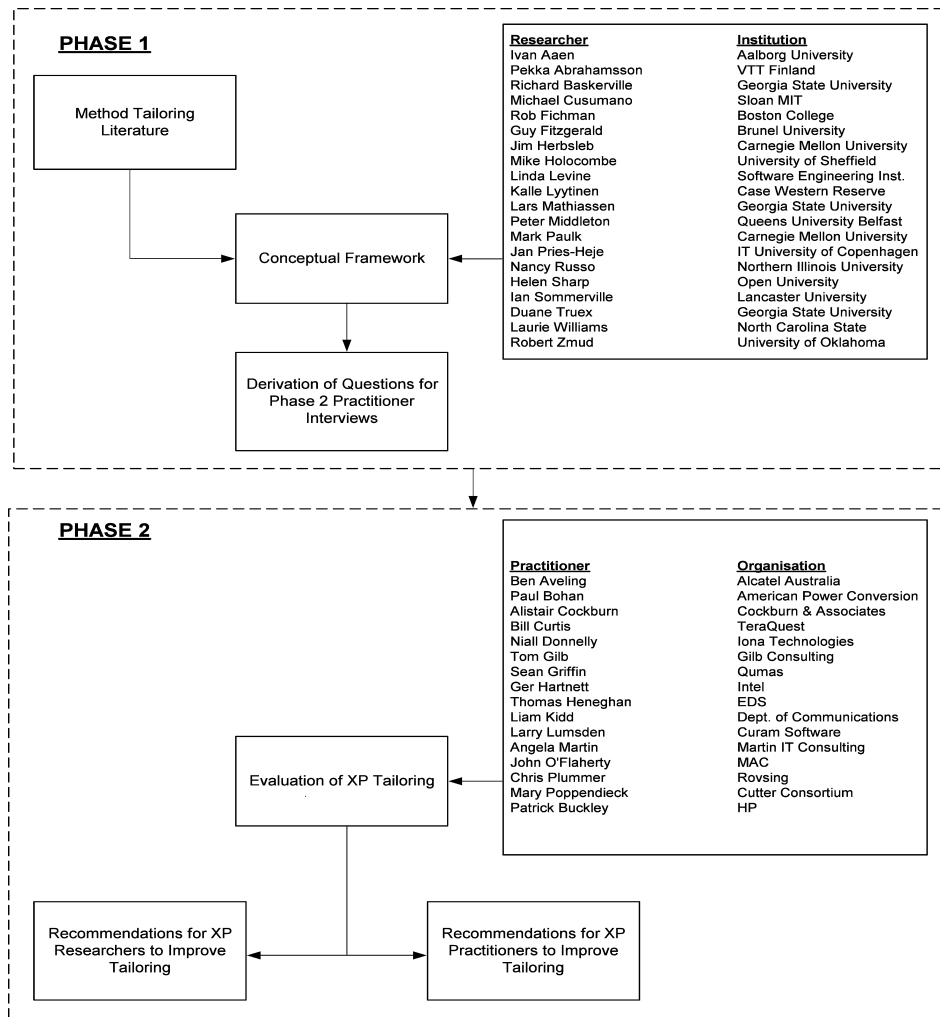| Practitioner	Organisation
Ben Aveling	Alcatel Australia
Paul Bohan	American Power Conversion
Alistair Cockburn	Cockburn & Associates
Bill Curtis	TeraQuest
Niall Donnelly	Iona Technologies
Tom Gilb	Gilb Consulting
Sean Griffin	Qumas
Ger Hartnett	Intel
Thomas Heneghan	EDS
Liam Kidd	Dept. of Communications
Larry Lumsden	Curam Software
Angela Martin	Martin IT Consulting
John O'Flaherty	MAC
Chris Plummer	Rovsing
Mary Poppendieck	Cutter Consortium
Patrick Buckley	HP

Fig. 2.   The research approach.

XP contains many different concepts, philosophies, tools, and practices [Beck 2000; Beck and Andres 2004]. In this study, XP was implemented in many different ways across the 16 projects contexts. In addition, many aspects of XP are quite tacit and intangible, and so measuring adherence to the norm is quite challenging. For all these reasons, understanding if and how XP practices are tailored would have been a very unwieldy and complex task without an appropriate structuring mechanism.

The rest of this section discusses the approach used to identify and select participants and to collect and analyze the data.

2.1 Selection of Expert Participants

This study involved a set of interviews with experts in the field of software development, an approach that is highly beneficial for applied research [Dalkey

Table I. Classification of Experts and Listing of Participants

Desired Background or Skillset	Method of Expert Identification	Minimum Selection Criteria
(1) Practitioners who have used agile methods	—Membership of relevant agile method groups (Agile Alliance, DSDM Consortium etc) —Personal contacts	>Five years' agile method experience >Three years' agile project management experience
(2) Practitioners who have worked in software development, and are aware of agile methods	—Membership of relevant societies (ITAA, Cutter Consortium etc) —Personal contacts	>Seven years' software development experience >Five years' project management experience
(3) Academics who have researched agile methods	Literature review of relevant academic and practitioner journals and conferences	≥Three agile method publications in refereed journal/conferences
(4) Software development researchers who are aware of agile methods	Literature review of relevant academic and practitioner journals and conferences	≥Five software development publications in refereed journal/conferences

and Helmer 1963; Linstone and Turoff 1975; Moore 1987]. First, combining the judgment of a large number of experts offers a better chance of getting closer to the truth. Second, it is easier to understand phenomena by obtaining the views of the actors. Given the ambiguous interpretation and use of agile methods and the fact that they are socially oriented methods [Beck 2000; Schwaber and Beedle 2002; Koch 2005], this is highly relevant in the context of this study. Finally, pooled intelligence is often suited to the resolution of complex and ill-defined problems [Dalkey and Helmer 1963], difficulties which typify the use of agile methods, and indeed the study of software development and the study of agility across all disciplines.

Group size theory varies in its suggestions regarding the ideal number of expert participants in such a study. Some general rules-of-thumb indicate five to 10 people for a homogenous population, but 15 to 40 people for a heterogeneous population, that is, people coming from different social and professional stratifications such as academics and practitioners, as is the case in this study [Delbecq et al. 1975; Uhl 1983]. This study involved 36 interviews, a figure at the upper range of the recommended group size.

Verifying expertise is somewhat difficult as it can be judged by status, experience, or "a myriad of other things" [Brown 1968]. A methodical selection of participants or allowing every willing person to take part is considered highly unscientific [Sackman 1975; Clayton 1997], and so systematic classification and selection were conducted. The skills and background of experts required for this study are listed in Table I, along with the basis for identification and selection. The minimum selection criteria were based on reasonable expectations as to the typical characteristics of a software development expert, and the criteria usually recommended for expert studies (e.g., Brown [1968]; Meyer and Booker [2001]). As well as selecting a mix of practitioners and academics, the selection process also ensured that at least half of the participants had the

experience of using or researching traditional, preagile methods, so as to enable comparison and critical reflection. It is also worth noting that the minimum criteria were lower in relation to the selection of agile-oriented practitioners and academics, as more stringent criteria requiring more industry experience or a large number of agile method publications are somewhat unrealistic given that these methods have such recent origins. The list of participating researchers and practitioners was presented in Figure 2.

2.2 Data Collection

A critical issue regarding the data analysis phase concerned the unit of analysis, particularly where the practitioner responses were concerned. At the time of the interviews, many of the practitioners were involved in a number of projects or roles. This was particularly true of those working as consultants or within a consulting organization. As the unit of analysis was at the project level, all practitioners were asked to consider a single project where they were "substantially" involved. The qualifying questions for this ensured that the project was of sufficient duration (>3 months), that the interviewees were involved to a significant degree (>60% of their time), and that they had a role on the projects which provided them with an informed opinion on the implementation and tailoring of XP on their projects (consultant, project manager, team lead, or developer).

Data was collected through personal face-to-face interviews, which is considered the superior data-gathering technique for qualitative studies such as this [Yin 2003]. Personal interviews are also well suited for exploratory research because they allow expansive discussions which illuminate additional factors of importance [Oppenheim 1992; Yin 2003]. Also, the information gathered is likely to be more accurate than information collected by other methods since the interviewer can avoid inaccurate or incomplete answers by explaining the questions to the interviewee [Oppenheim 1992].

A guiding script was prepared for use throughout the interviews to establish a structure for the direction and scope of the research. It also ensured coverage of all aspects of the study with each respondent, and helped achieve some element of distance between the interviewer and interviewee, while permitting the researcher to compare and contrast responses [McCracken 1988]. The interview questions were circulated in advance to allow participants to consider their responses prior to the interviews. The questions were largely open ended, allowing respondents to convey their experiences and views on the socially complex contexts that underpin software development and agile method use [Oppenheim 1992; Yin 2003].

The interviews lasted between 50 and 120 min (average = 85 min). The interviews were conducted in a reflexive manner, allowing the researcher to follow up on insights uncovered midinterview and to adjust the content and schedule of the interview accordingly [Trauth and O'Connor 1991]. Furthermore, a diary was kept of questions asked during each interview and their effectiveness, and refinements and additions were made to the set of questions prior to the next interview. To aid analysis of the data after the interviews, all were

recorded with each interviewee's consent, and were subsequently transcribed (total 470 pages), proofread, and annotated. In any cases of ambiguity, clarification was sought from the corresponding interviewee, either via telephone or email.

2.3 Data Analysis

For data analysis, we adopted coding procedures recommended for qualitative research, systematically labeling concepts, themes, and artifacts so as to be able to retrieve and examine all data units that referred to each issue across the interviews. The coding structure adopted in this research consisted of three distinct mechanisms. First, an identification code was attached to each piece of text extracted from a transcript (R1…R20 for researchers and P1…P16 for practitioners) to ensure participant anonymity. Second, a classification schema was built, acting as what Miles and Huberman [1999] called a set of "intellectual bins," so as to segment and filter the interview data collected. Finally, pattern coding was used to "identify emergent themes, configurations or explanations" [Miles and Huberman 1999, p. 47].

3. FINDINGS

3.1 Phase 1—Development of the Conceptual Framework

In this section we present a description of the method and developer characteristics which facilitate effective method tailoring. These emerged from a review of the literature and interviews with the senior researchers. Each section includes a brief synopsis of the literature underpinning each characteristic, and also includes a review of any relevant XP literature pertaining to that characteristic. This is completed by the proposed conceptual framework encapsulating these characteristics.

3.1.1 *Method Characteristics That Facilitate Tailoring.*

3.1.1.1 *Explicit Statement of Method Boundaries.*   The method author can specify boundaries describing under what conditions the method should or should not be used. This allows teams to filter out potentially unsuitable methods when choosing which approach to employ [Brinkkemper 1996].

In relation to XP specifically, it has been suggested that there are environments unsuitable for its use, such as small teams and distrusting customers [Beck 2000]. However, the literature contains many studies of its nonconventional use (e.g., large teams [Bowers et al. 2002; Crispin and House 2003; Cao et al. 2004], startups [Auer and Miller 2002], distributed development environments [Kircher et al. 2001; Stotts et al. 2003], greenfield sites [Rasmusson 2003], educational environments [Johnson and Caristi 2003; McDowell et al. 2003; Wainer 2003], open source development [Kircher and Levine 2001], and systems maintenance [Poole and Huisman 2001]). This suggests the boundaries of XP use are not so clearly identified, nor indeed accepted.

An analysis of the proprietary[4] texts accompanying many other agile methods reveals a similar tendency among method authors not to state method boundaries and limitations. Schwaber and Beedle [2002] were adamant that "Scrum works for all projects" [p. 112] regardless of size, system type, system criticality, or expectations regarding quality. Cockburn [2001] conceded the limitations of Crystal, but then included different variants to widen its applicability. While there have been caveats mentioned in relation to other methods [e.g., Stapleton [1997]; Poppendieck [2001]; Ambler [2002]; Coad and Palmer [2002]), an explicit statement of method boundaries is missing for them all. One explanation for this is the possibility that these methods are indeed applicable in all circumstances, unlike their traditional counterparts. However, such one-size-fits-all application of agile methods has been questioned in several research studies [e.g., McBreen [2003]; Stephens and Rosenberg [2003]; Koch [2005]).

3.1.1.2 *Contingency Built In to Method Itself to Guide Tailoring.* To aid tailoring, a method can contain what Iivari [1989] called *built-in contingency*, whereby the method itself provides guidance for the tailoring process, containing an encompassing framework allowing it to be adjusted to fit any context. While some traditional methods acknowledge the need for such flexibility [e.g., Wood-Harper et al. [1985]; Booch [1994]; Coleman et al. [1994]), very few include mechanisms to facilitate such tailoring [Avison and Wood-Harper 1991].
From an analysis of the literature, it seems that XP does not adequately deal with this issue either. In comparison with other methods, XP's practices are "highly prescriptive" [Abrahamsson et al. 2002] and binary; either the practices are followed or they are not [McBreen 2003]. Although Beck and other XP enthusiasts have acknowledged that tailoring can and indeed should be conducted, Crystal [Cockburn 2001] is the only agile method which builds contingency into itself, offering clear instructions as to how tailoring should be accomplished. This is done by including variants to be used depending on project needs. The lack of guidance regarding how XP can be tailored is reminiscent of the older, more traditional methods where the need for flexibility is acknowledged but not addressed.

3.1.1.3 *Clear Description of Method and Rationale Behind Method Practices.* Of the various method engineering frameworks in existence, almost all require an explicit rationale to guide the use of method fragments (e.g., Tolvanen and Lyytinen [1993]; Cronholm and Goldkuhl [1994]; Harmesen et al. [1994]; Brinkkemper [1996]; Harmsen [1997]). This provides the software development team with as much information as possible about each method fragment, and specifically why it should be included in the method being constructed.
Current XP literature which specifically addresses the rationale behind each of its constituent practices is scarce and what does exist is inconclusive. On

---

[4]In this article, the term *proprietary texts* is used to refer to those works in agile method literature commonly accepted as originating handbooks, or official guides of a method; for example, XP's proprietary texts would be Beck [2000] and the later Beck and Andres [2004], while that for Scrum guides would be Schwaber and Beedle [2002] as the most noted.

the one hand, there are a couple of dissenting texts which have portrayed XP as being irrational and vague [McBreen 2003; Stephens and Rosenberg 2003]. In addition, several studies have also highlighted the fact that some XP practices are nonprescriptive, and represent a high level of abstraction, lending themselves to inconsistent interpretation and implementation [Abrahamsson et al. 2002; Boehm and Turner 2004; Koch 2005]. On the other hand, however, an analysis of the literature reveals numerous texts dedicated to XP, and the method is also predominant in most research on agile software development. In many of the texts, a whole chapter is dedicated to each XP practice. Therefore, from an analysis of these texts at least, it is hard to argue that the creators of these methods have not provided an adequate explanation of their constitution and rationale.

3.1.1.4 *Independence of Individual Method Practices.* A further key attribute which renders a method amenable to tailoring is the extent to which its individual component practices are independent, allowing them to be separated or combined without fear of unknown subsequent effects [Kumar and Welke 1992].

Some XP literature has suggested that agile method practices can be tailored with ease, and there are numerous cases to support this notion [e.g., Bowers et al. [2002]; Rasmusson [2003]; Cao et al. [2004]). Beck and Fowler [2001] stated that "no two XP projects will ever act exactly alike" [p. 52], and once a software development team is comfortable with the basic process, it can change the practices to fit the context more precisely. However, many have argued that one of the most distinctive features of XP is that its practices are not independent, but instead are very "tightly coupled" [Auer and Miller 2002], "daisy-chained" [Stephens and Rosenberg 2003] "interdependent" [Beck 2000], and "synergistic" [Martin 2003]. Beck [2000, p. 12] stated that "any one [XP] practice doesn't stand well on its own . . . and they require the other practices to keep them in balance," although he later claimed that the practices within the modified version of XP are more independent [Beck and Andres 2004]. Boehm and Turner [2004, p. 115] cited an unnamed "agilist" who rejects any partial use of XP and claimed that "the pieces fit together like a fine Swiss watch." Despite the fact that XP is supposedly adaptable to a wide variety of projects, Stephens and Rosenberg [2003, p. 77] stated that the "authors have got it exactly the wrong way around." Instead, they likened XP to a "self-referential safety net," where, even if some practices add no value, it is impossible to remove them if they are necessary to hold the other ones in place.

3.1.2 *Developer Practices That Facilitate Tailoring.* While there are a number of ways in which the method itself can aid the tailoring process, much responsibility also lies with developers to tailor effectively. Key developer practices are now described and accompanied by an analysis of the XP literature.

3.1.2.1 *Identification of Project Context Dependencies.* In a contingency-based tailoring process, a method is selected from a portfolio of alternatives, matching the method to the project context. The process of mapping the characteristics of the project facilitates a more informed process of method

selection. The explicit identification of the traits and nuances of the development environment improves the chances of achieving a close alignment between method and the environment [Kumar and Welke 1992; Harmsen et al. 1994; Brinkkemper 1996]. Many studies have proposed features, or "situation dependencies" [Kumar and Welke 1992] of a software development project environment which should be taken into consideration by developers when making this decision (e.g., Davis [1982]; Gremillion and Pyburn [1983]; Iivari [1989]). These characteristics can be categorized broadly as technical (i.e., type of system or programming language), organizational (i.e., development culture or maturity), or human (i.e., level of experience), among others.

In terms of XP, researchers have proposed sets of situational characteristics upon which the decision to adopt XP and other agile methods should be based (e.g., Boehm and Turner [2003, 2004]; Koch [2005]). These include team size, relationship with the customer, criticality of the system, dynamism of the environment, developer competency, team culture, and preexisting tools and processes. However, the XP literature sheds little light of the extent to which developers go through this process and choose XP on the basis of situational characteristics—neither Boehm nor Koch indicated whether their models are used in practice. There is no rigorous research as far as we are aware that examines the extent to which developers identify which dependencies apply to their project and whether they are basing the decision to use or not use XP based on these dependencies.

The literature suggests that a lot of adaptation and extension to the XP method may be required when applied to, for example, large teams [Bowers et al. 2002; Crispin and House 2003; Cao et al. 2004], startups [Auer and Miller 2002], distributed development environments [Kircher et al. 2001; Stotts et al. 2003], greenfield sites [Rasmusson 2003], educational environments [Johnson and Caristi 2003; McDowell et al. 2003; Wainer 2003], open source development [Kircher and Levine 2001], and systems maintenance [Poole and Huisman 2001].

3.1.2.2 *Familiarity with a Range of Methods and Method Fragments.* To support effective contingency-based selection, the software development team should be familiar with a broad range of methods from which to choose. This requirement is often cited as a key limitation of contingency theory, given that most developers have experience of no more than one or two methods, and a comprehensive knowledge of even one full method is rare [Kumar and Welke 1992]. Familiarity also has a key role to play in method engineering, as developers should have an understanding of many methods and method fragments in order to construct new methods and combinations of fragments [Brinkkemper et al. 1998].

While there are numerous case studies of XP use (see previous section), none, as far as we are aware, identified the extent to which the teams had knowledge or experience of other methods, agile or otherwise. Almost all XP case studies have focused on one project setting, and while these have revealed many insights, they have not revealed whether XP was selected from a pool of methods, or whether developers were even familiar with any other methods.

A small number of studies such as Fitzgerald et al. [2006] have described the amalgamation of XP and Scrum in a single project but this is certainly not enough to conclude that many teams have had similar combined knowledge.

3.1.2.3 *Disciplined and Purposeful Approach to Method Tailoring.* As stated in the introduction to this article, method tailoring is so pervasive in software development that the textbook implementation of methods in practice is rare. However, such efforts are often ad hoc, suboptimal, and problematic. Tailoring is still carried out in an ad hoc fashion, and little is learned about tailoring across projects [Iivari 1989; Brinkkemper 1996]. There are a number of frameworks for engineering and tailoring methods, some of which classify method fragment attributes (e.g., Brinkkemper et al. [1998, 1999], while others classify the goals and values to which method fragments should make a contribution (e.g., Ågerfalk and Wistrand [2003]; Ågerfalk and Fitzgerald [2005]). According to these studies, the team should use these to categorize fragments, and structure the development of a new method, ensuring the fragments link cohesively and contribute to a common goal.

An analysis of the XP literature suggests that many efforts have been made to tailor XP to suit a variety of contexts, as outlined above. These studies have usually provided a detailed account of the tailored and enacted practices and in some cases the relative success or failure of the initiatives. However, there has been little focus specifically on the extent to which such tailoring is done in a disciplined manner, and it is not known whether or how teams evaluate each XP practice before deciding whether to adopt it, tailor it or remove it.

3.1.3 *Summary of Phase 1.* This section has outlined the key developer and method characteristics which can aid the tailoring process, resulting in a method which provides as close a fit to the project context as possible. We summarize the results of this section in Figure 3 and propose that the method and developer characteristics identified improve the effectiveness of method tailoring.

3.2 Phase 2—Assessment of XP Tailoring in Practice

In the first phase of this study, we derived a conceptual framework (Figure 3), identifying the key method and developer characteristics that contribute to effective method tailoring. This phase also identified a realtive paucity of existing research examining the extent to which XP as a method and developers using XP exhibit these characteristics. We now apply the framework in Phase 2 to evaluate the extent to which XP and developers using XP exhibit the characteristics which contribute to effective method tailoring. The 16 practitioners interviewed were listed earlier in Figure 2. To protect anonymity the respondents have been assigned the pseudonyms P1 to P16 in a random order.

3.2.1 *Method Characteristics That Facilitate Tailoring.*

3.2.1.1 *Explicit Statement of Method Boundaries.* The earlier review of the literature identified a lack of defined boundaries as to where XP could and could

Method Characteristics

- Explicit statement of method boundaries - where it is/is not applicable
- Contingency built in to method itself to guide tailoring
- Clear description of method and rationale behind method practices
- Independence of individual method practices - success not dependent on synergy from application of multiple practices

Developer Practices

- Identification of project context dependencies
- Familiarity with a range of methods and method fragments
- Disciplined and purposeful approach to method tailoring

improves effectiveness of
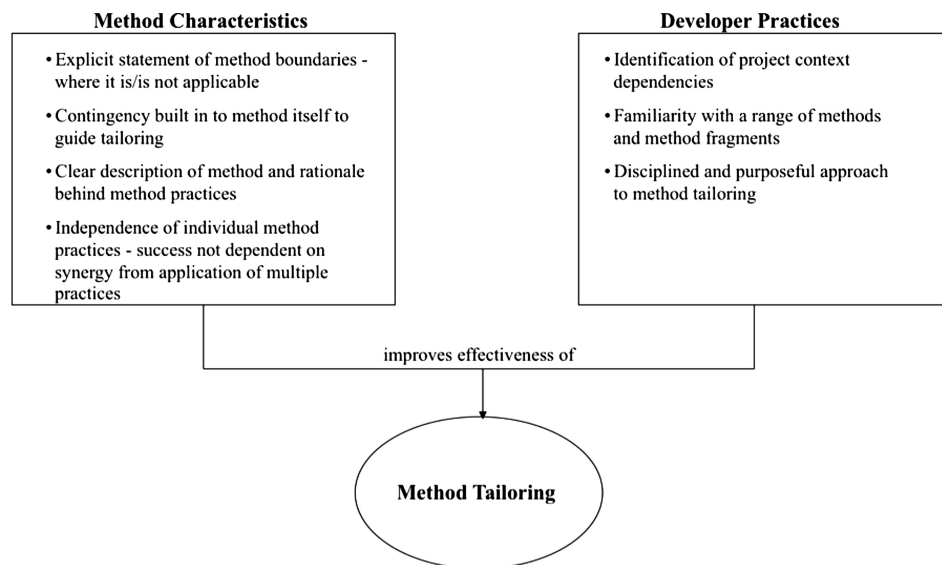
Method Tailoring

Fig. 3.   Method and developer characteristics contributing to method tailoring.

not be applied. When questioned, 14 of the 16 practitioners stated that, for the purposes of their project, they too had trouble identifying the boundaries of XP suitability. In seven of the 16 project teams, members read XP documentation or attended conferences with the intention of determining the extent to which they could apply XP given the specific context of their respective projects, but none felt their efforts made this decision sufficiently clear. As one stated, "I heard many interesting opinions but they were inconsistent, conflicting, and when I left I wasn't really any clearer on what practices we could or couldn't use" (P7). In one case the practitioner in question (P8) organized a "birds-of-a-feather"[5] session at an agile methods conference entitled "When does XP not work?" The purpose of the session was to "get past the vague responses and figure out once and for all whether XP would work on our project." According to this practitioner, 35 people attended and although all seemed to consider this issue to be crucial, "there was no consensus on anything—we left no wiser than when we arrived" (P8).

On all three projects where external XP consultants were used, there was reluctance among these consultants to draw such a boundary and declare any XP practice unsuitable for use. Instead, all three recommended a trial period where the team would reflect on all practices before making a decision regarding their use. This was described by one practitioner:

> He [the consultant] would not commit when we asked him to advise us on which
> XP practices would and would not work. With us he insisted on a "try first"

---

[5]Birds-of-a-feather session: a scheduled slot at a conference where individual attendees can post topics for like-minded individuals to discuss.

> approach where all practices are tried and only dropped if not working. But
> after 6 months and his refusal to accept any arguments against the method,
> I'd say his philosophy was "try first, and if it doesn't work then just try harder."
> (P7)

In the course of their projects, many of the practitioners encountered consultants or XP advocates who referred to the method as the "bible" (P4, P7, P11, P12) with something almost akin to religious fervor in their view that the method should never be tailored or even questioned. One practitioner recalled the experiences of working with one consultant whom he called an "agile apostle," where the practices of XP were continuously referred to as "the 12 commandments" (P4).

These findings support the earlier analysis of the literature—the boundaries of XP application are unclear. XP texts and XP consultants are often slow to concede the limitations or boundaries of the method.

3.2.1.2 *Contingency Built In to Method Itself to Guide Tailoring.*   As shown by the sporadic and diverse adoption of practices across the 16 projects studied, it is clear that in all cases XP was tailored to some degree. However, the literature suggests that a highly tailorable method should guide some or all of the tailoring process. The 16 practitioners were asked to consider the extent to which XP texts, their research, and other available XP resources contributed to their tailoring efforts. The analysis revealed that 15 practitioners had read a significant number of XP texts and got information from other sources such as blogs and research articles, and in fact 14 of the 16 had attended at least one XP conference. In another instance the project was a small part of a larger initiative, and so all tailoring decisions were made at a higher level. This leaves 14 relevant projects for this part of the study.

Assessing how reading an XP book, a tailoring process document, or other material impacts the tailoring decision process is very difficult, as the association may be informal and subconscious. Nevertheless, across the 14 projects, none of the practitioners thought that the XP method and associated material guided the tailoring process in any meaningful way. Instead, they stated that tailoring efforts were based on the intuition and personal experiences of the managers or developers involved.

The earlier discussion of the literature suggests that practitioners rarely pay attention to method documentation in practice, and this could help explain why such documentation did not impact tailoring efforts. However, in this study, the responses indicate that, rather than practitioners being ignorant of XP material, many actually sought assistance from such resources, but to no avail. To illustrate, one practitioner described his longing for a simple document accompanying an agile method, telling him "if a particular circumstance exists, do these steps, if a different circumstance exists, do these steps, and so on" (P5). Another practitioner's experience illustrated a search for help, but more significantly showed that the lack of built-in contingency had a real negative impact on the project:

> We changed a lot of things about XP. It took a long time to perfect, given we were
> flying in the dark, on a trial and error basis, but we got there. And I think we are

> more agile. I just wish the option to use these alternatives could have been part of the method. It would have saved a lot of time, effort and uncertainty. (P1)

The absence of built-in contingency is not specific to XP, or indeed to agile methods in general, as this has been a problem associated with traditional methods at least as far back as the 1980s [Iivari 1989]. However, it is of more concern in an agile method context, given that, as discussed earlier, a method that claims to be agile should be as adaptable and amenable to tailoring as possible.

3.2.1.3 *Clear Description of Method and Rationale Behind Method Practices.* All 16 respondents stated that they had a clear understanding of XP, at least at a high level. This is not surprising given that a key criterion in the selection of participants in this study was extensive knowledge and experience of XP. The interviews did, however, reveal a number of additional issues regarding a lack of clarity in the rationale behind XP practices.

First, while the practitioners themselves had a good understanding of XP, some people in their respective teams found certain XP practices quite difficult to grasp. Implementing some practices such as *pair programming* and *on-site customer* was relatively straight-forward. However, particular practices, such as the *system metaphor*[6] and *simple design* practices, were not. A common complaint was that practitioners found the level of abstraction across XP practices to be quite varied, making it difficult to make a rational tailoring decision. One described practices including *pair programming* as "prescriptive, operational and detailed," but practices such as the *simple design* and *metaphor* as more "abstract" and open to wider interpretation (R3).

In addition to a lack of clarity regarding how some XP practices should be implemented, the study indicates that the rationale behind some agile method practices is not that clear, and, as one practitioner stated, "Unless you understand the rationale, you can't make an informed decision about extending that step, tailoring it, [or] dropping it" (P10). Again, the *system metaphor* and *simple design* practices were identified as being problematic, with 11 of the 16 practitioners stating that the rationale behind these were unclear to them and/or their respective teams. As stated earlier, the emergence of most agile methods has been accompanied by proprietary texts which clearly describe the rationale behind the method's practices (e.g., Beck [2000]; Cockburn [2001]; Schwaber and Beedle [2002]). However, one respondent suggested the perceived lack of clarity regarding these practices could arise as a result of these methods being communicated second and third hand without the aid of proprietary documentation.

3.2.1.4 *Independence of Individual Method Practices.* All 16 practitioners stated that tailoring of XP was made more difficult by the fact that many practices, rather than being independent, are actually highly interdependent and tightly coupled. Various participants recalled problems or concerns when

---

[6]The system metaphor practice has been removed in the most recent version of XP [Beck and Andres 2004], but featured in the version adopted by the 16 projects studied.

trying to separate XP practices on their respective projects, and described them as being "interrelated" (P1, P7), "interconnected (P9), "fused" (P11), "meshed" (P8), "knitted" (P16), "tightly coupled" (P4), or "tethered" (P12), which together form a set of "checks and balances" (P6).

Most practitioners had experience of using traditional methods alongside or prior to adopting agile. When asked to compare independence of XP practices to that of other methods, the consensus was that a lack of independence was a problem irrespective of the method being used. All the practitioners stated that, in the experience of their current or most recent XP project, this issue was exacerbated by the "softer" (P1), "social" (P3) nature of many of the XP practices. This rationale was captured quite well by one interviewee:

> When I am thinking about asking the team to write a document or use a tool, I know what [subsequent] effect it will have on the other work they are doing. But with XP it's a mess. I don't know if pairing them off or getting them to present their work twice a day is going to help everything else or cause it to implode. (P8)

Recognizing that XP practices were perceived to lack independence, the interviewees were asked if they "grouped" or "clustered" any interrelated XP practices on their projects. While many acknowledged that XP practices could and perhaps should be grouped into clusters, none identified such practice interdependencies or explicitly attempted this exercise on their project. Even the three interviewees who felt that certain parts of commercial agile methods could be decomposed still conceded that there were "clusters of practices" embedded within XP that are so reliant on each other that they should not be decomposed and applied in isolation.

An analysis of the projects also showed that, not only were XP practices lacking independence, but that this was having a negative impact in practice. Many wanted to remove what they perceived to be non-value-adding practices, but were reluctant due to the embedded nature of these practices and uncertainty about what impact their removal would have on the effectiveness of other practices. As a result, many XP practices were retained even though they added little or no value and in some cases even had a negative impact on the project. This problem is somewhat reminiscent of Stephens and Rosenberg's [2003] critique of what they called the *self-referential safety net*, whereby no fragment can be removed regardless of its limitations, due of the other fragments which are dependent on it.

3.2.2 *Developer Practices That Facilitate Tailoring*

3.2.2.1 *Identification of Project Context Dependencies.* The literature suggests that developers should consider situational dependencies when deciding on which method to adopt. Of the constructs being examined in this article, this task is perhaps the most difficult, given that the decision to base method selection on its suitability to the project environment is rarely explicit.

When asked to reflect on their most recent XP projects, only one practitioner (P5) said that a formal method selection process had been followed, where project dependencies were explicitly identified and used as a basis to identify XP

as the most suitable method. In this case, the practitioner read a number of XP and agile method texts to determine the types of projects most suited to XP. The most valuable of these, according to the practitioner, was Boehm and Turner's [2003] assessment framework, and this became a core part of the evaluation process. All 15 of the team's developers, an on-site customer, and three other stakeholder representatives met on three occasions to identify where their team and project were positioned along each of the Boehm and Turner framework's five axes. This represents a good working example of method selection based on the identification of situational dependencies.

None of the other 15 projects identified situational dependencies in such a clear and transparent manner. While a few of the other practitioners stated that they would have informally considered such dependencies themselves, the majority stated that the characteristics of their projects had very little impact on the decision to adopt XP. This is supported by the fact that none of the 16 projects displayed all of the characteristics typically associated with XP. For example, 11 involved distributed development, nine involved large teams, nine could not facilitate an on-site customer (or a customer did not exist), eight were required to comply with organization-wide development processes or reporting structures, and four involved critical systems.

In many cases the decision to adopt XP was often driven by one single "champion" without input from any other team members or stakeholders (P1, P2, P3, P4, P6, P8, P10, P16). In such cases, the decision to adopt XP was made first, with the suitability to context being an "afterthought" (P4). On the other hand P5 and P6 recalled exemplory cases of collective involvement in the decision-making process. In P5's case, a series of three semiformal workshops on XP were conducted which all team members attended and where they raised any potential concerns and issues. These were then discussed and either resolved the same day, or members of the team were appointed to research the issues, get external advice, and try to identify a resolution. Regarding P6, all nine team members attended at least one XP conference to familiarize themselves with the method and to ensure they were satisfied with continuing to use XP.

In cases where the project characteristics did not suit XP, rather than dropping the method, there were multiple examples of "lubricating" (P3) the environment to suit XP. For example, large teams were broken down into smaller teams (P6, P8, P11, P12, P16), a proxy customer was used in the absence of a real customer (P1, P2, P4, P8, P16), and systems were decomposed into critical and noncritical components (P8, P10). While tailoring the environment to suit XP seemed very effective in certain circumstances, some interviewees cited examples where this could be taken to an unrealistic extreme. For example, P16 complained that one agile method consultant persistently recommended the use of XP for a government project despite its poor fit, and "couldn't grasp the fact that the structure of 50,000 people can't revolve around what suits one 10-person team" (P16).

3.2.2.2 *Familiarity with a Range of Methods and Method Fragments.* Contingency literature suggests that method users should ideally be familiar with a range of methods and fragments to allow the effective tailoring and substitution
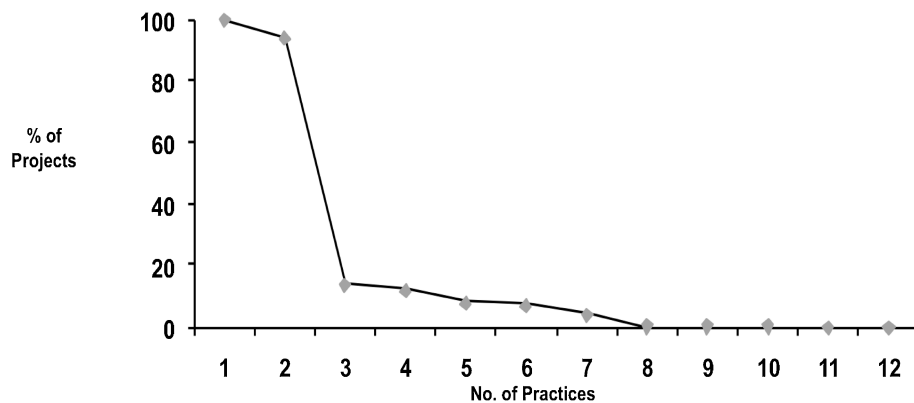
Fig. 4. Sporadic adoption of XP practices.

of fragments. Therefore, even though XP was used on all 16 projects' studies, ideally the developers should be familiar with a range of other methods.

While all 16 developers interviewed had some level of textbook or second-hand knowledge of other agile methods apart from XP, this was restricted to Scrum in 15 cases, and knowledge of any other methods was very sporadic. In terms of practical, hands-on experience, only three of the 16 practitioners had used an agile method other than XP, and interestingly again, all of these involved a combination of XP and Scrum. In fact, every interviewee had more practical experience of traditional method usage than use of other agile methods with the exception of XP. The primary reason for this was that, prior to adopting XP, these developers used either a traditional or an amethodical approach to software development. In all 16 cases, once the project team had committed to a transition to agile, the first point of departure was XP. No team had moved from one agile method to another. Previous research has found that developers using plan-driven methods are usually not familiar with a portfolio of methods and fragments and are therefore not in a position to tailor methods successfully using a contingency approach [Kumar and Welke, 1992]. Our findings in this study suggest that developers using XP are no more likely to be familiar with a range of methods, and where they are familiar with alternatives, these are more likely to be traditional than agile.

An analysis of responses suggests that even hands-on familiarity with XP in its entirety was rare. According to an analysis of interviewee responses, while software development teams may say they are using the method, in many cases only a minority of practices are actually implemented. As illustrated in Figure 4, while all teams used at least one XP practice, none used more than 10 XP practices. In fact only 25% of those projects were adopting more than half of the practices.

3.2.2.3 *Disciplined and Purposeful Approach to Method Tailoring.* From the data collected it is clear that many of the tailoring efforts were conducted in an ad hoc, unstructured manner. In most cases there was little evidence to suggest that due consideration was given to each practice before discarding it.

In fact, rather than starting with 12 practices and reducing the number actually implemented based on some rationale, most projects adopted an incremental approach, starting with a few practices but never actually getting beyond a few key ones. This is directly at odds with recommendations in the method tailoring literature which suggests that developers trial the "vanilla" version of the method or at least evaluate each of the method's practices before deciding which to adopt, tailor, or remove. Only one practitioner stated that, on his project, the "pros and cons" of each practice were debated thoroughly before deciding whether to adopt it or not (P15).

There was also evidence to suggest a possible correlation between familiarity with practices and their selection. On many projects practices were retained if they were perceived to be easy or if they had been used on previous projects. In contrast, however, many of the practices which were not used were ones where the team had difficulty understanding them. As one practitioner stated: "The practices the [developers] knew absolutely nothing about were just the ones they loved to drop" (P12).

Once each project commenced and a decision was made regarding what practice to use, there was little explicit monitoring of ongoing adherence to the XP method. In most instances practices were only carried out if "the developers felt like it" (P1). P15 was the only exception, where developers had to present their adherence or nonadherence to each key practice at every postiteration retrospective meeting, and had to explain the reasons for any noncompliance. Ongoing tailoring usually was not centrally controlled by management, but was instead left to the discretion of the teams. In some cases this was because the project managers felt that developers were the most appropriate people to make the decision:

> I could have forced them to use practices, but isn't developer empowerment the whole idea behind XP? Anyway, if they don't think it's worth doing, then I'm not going to tell them otherwise. (P11)

This is in line with the values of XP where developer empowerment is encouraged and viewed positively.

While there may be benefits to allowing ad hoc tailoring, disciplined tailoring is nevertheless a key aspect of effective method tailoring. Our study provided further evidence to suggest that ad hoc tailoring had negative effects for the projects studied. First, in some cases, tailoring was ad hoc not because the respective managers decided to empower the developers but simply because ensuring compliance with XP practices was beyond the control of the managers:

> XP is not like other methods. I can get the team to carry out technical procedures; but the social side of XP that goes with pairing, standups and constant collaboration—if they don't want to do it, I can't make them. (P3)

Second, unstructured tailoring always resulted in projects being subjected to what the practitioners called the "haphazard" (P12), "disjointed" (P6), "sporadic" (P3), or "patchy" (P15) use of XP. This does not refer to adherence to selected practices as illustrated in Figure 3. Instead, the interviewees were referring to differing levels of adherence within the team itself, for example,

some developers using some practices and not all developers using the same practice in the same manner.

These findings suggest that problems cited by previous researchers (e.g., Iivari [1989]; Kumar and Welke [1992]) in relation to traditional methods may still hold true for XP; tailoring is still left to the discretion of developers, it is still carried out in an ad hoc fashion, and little is learned about tailoring across projects. The need for diligent tailoring should perhaps be considered as even more of an imperative, according to one researcher in the first round of interviews, given the "uncharted" nature of XP (R5). He argued that some of the flawed parts of older methods have been exposed through years of application, but that it was "inexcusable" for a team to discard parts of XP when so little is still known about its use.

4. CONCLUSIONS

This article contributes to XP and general software development literature by introducing a conceptual framework of effective method tailoring from two complementary perspectives: that of the characteristics of the actual method being tailored, and that of the characteristics of the individual developers responsible for method tailoring. This rich, two-faceted, research approach has not been a feature of previous research on the method tailoring topic. The framework was then used to analyze XP and the use of XP in practice. In contrast, the existing literature on XP and other agile method tailoring tends to be descriptive rather than grounded in a theoretical framework or set of principles. In fact, these studies usually describe the final tailored method and its success, but do not discuss the tailoring process itself in any significant detail.

We considered the extent to which XP fulfils each of the four method characteristics—the explicit statement of method boundaries, contingency built in to the method itself to guide tailoring, a clear description of the method and rationale behind method practices, and the independence of method practices. We found these to be largely absent in XP, which is surprising given that, as stated earlier, a method labeled *agile* should logically contain these agile properties.

It is also clear from the study that there are a number of deficiencies which hinder XP tailoring in practice. Developers who use XP are not adhering to the tailoring "best practices," identified in this study based on interviews with senior researchers and a review of the extant method tailoring literature. There are improvements that could be achieved in relation to each of the developer practices, and doing so would improve the effectiveness of future tailoring efforts. Identifying project context dependencies, ensuring familiarity with more than one agile method, and adopting more structured tailoring approaches would lead to more effective XP tailoring efforts than exist at present.

In Tables II and III we provide a synopsis of findings in relation to the characteristics of XP relevant to method tailoring, as well as of practices for developers currently using XP. In Table II we summarize our findings in relation to method characteristics and identify a number of recommendations for software engineering research. Our study identified many gaps in the literature,

Table II. Recommendations for Software Engineering Research to Improve XP Tailoring

Construct	Finding	Recommendations for Software Development Researchers
Explicit statement of method boundaries	Fourteen of 16 teams studied had difficulty identifying where XP should and should not be applied. XP conference attendance and help from external consultants did not provide any substantial assistance with this issue. Opinions suggested that this was a problem throughout the XP user community.	(1) Determine levels of project success across different potentially problematic software development environments (e.g., distributed development, large teams, critical systems, inexperienced developers).
Contingency built in to method itself to guide tailoring	No practitioner thought that XP guided the tailoring process in any meaningful way, despite the fact that some developers needed and actively sought such guidance. All tailoring efforts were based on team members' own opinions and preferences.	(2) Identify alternatives for each XP practice, which achieve the same or similar goals and objectives (e.g., instant messaging, screen sharing, video conferencing, and common file storage can help replace XP's practice of colocation in a distributed development environment).
Clear description of method rationale behind method practices	There was mixed opinion regarding how clearly XP texts explain the rationale and execution of its underlying practices. Most were unclear as to the exact advantages and disadvantages of each practice, and were concerned that many accounts of benefits are often anecdotal, or too subtle to be clearly identified.	(3) Quantitative research to determine advantages/ disadvantages of XP practices. (4) In-depth qualitative research to uncover more subtle, softer advantages/ disadvantages of XP practices.
Independence of individual method practices	Problems or concerns regarding splitting of XP practices occurred in 10 of the 16 projects studied (P1, P4, P6, P7, P8, P9, P11, P12, P15, P16). The consensus was that the social and softer nature of XP practices makes it very difficult to identify codependencies and knock-on effects between practices. This had a negative impact in some cases. For example, P4, P5, P8, P11, and P12 wanted to remove non-value-adding or problematic practices but decided not to for fear of such unknown codependencies.	(5) Quantitative research to determine corelations between use of individual practices and (i) effectiveness of other practices and (ii) project success. (6) In-depth qualitative research to uncover more subtle, softer effects of use/ non-use of individual practices practice on (i) effectiveness of other practices and (ii) project success.
	Existing literature suggests that there are "clusters" of practices that are codependent, as opposed to simply pairs of practices. None of the project teams studied had managed to identify any such clusters.	(7) Quantitative research to determine corelations between use of groups or "clusters" of practices and (i) effectiveness of other practices within that cluster and (ii) project success.

Table III. Recommendations for Software Practitioners When Tailoring XP

Construct	Finding	Recommendations for Software Development Researchers
Identification of project context dependencies	Decision to adopt and tailor XP rarely involved a formal analysis of situational dependencies (1 of 16 projects studied—P5). The subsequent mismatch directly caused failure and future abandonment of the method in some cases.	(1) Conduct a formal analysis of method's suitability to the project environment (e.g., Boehm and Turner's [2003] analysis model was used very effectively by P5 for this purpose).
	Decision to adopt XP, as well as subsequent tailoring and implementation decisions, were often driven by one single "champion" without input from any other team members or stakeholders (8 of 16 projects studied—P1, P2, P3, P4, P6, P8, P10, P16). In some cases this resulted in a biased, uninformed decision without adequate consideration of negative consequences of introducing XP.	(2) Involve all developers and stakeholders in (i) the decision to adopt or not adopt XP, (ii) the tailoring of XP, and (iii) the implementation of XP (e.g., P15 held three semistructured, open-invitation workshops to identify and resolve various developer issues).
	Rather than tailoring XP to the environment, the organization or team was tailored, sometimes substantially, to suit the method (10 of 16 projects studied—P1, P2, P3, P4, P6, P8, P10, P11, P12, P16). However, this caused significant problems in some cases (e.g., P16).	(3) Identify any organizational or project "breaking points" (the maximum tolerable change), and ensure these points are not crossed, regardless of what XP requires (e.g., frequency of iterations, degree of collocation, average weekly working hours).
Familiarity with a range of methods and method fragments	Most developers using XP did not even have sufficient knowledge of all XP practices. As a result, the teams studied tended to implement easier practices and ignore more challenging ones. These practices were either omitted completely or implemented poorly as a result—only 25% of projects studied implemented more than 50% of the practices.	(4) Train developers on all XP practices (e.g., 1-day in-house tutorial, Web research).
	Developers' experiences of many XP practices were often second hand or textbook based, and insufficient for informed tailoring.	(5) Include practical, hands-on components in XP training (e.g., work shadowing, games, role playing (`http://www.xp.be/xpgame.html`), mentoring).

(*Continued on next page*)

Table III. (*Continued*)

Construct	Finding	Recommendations for Software Development Researchers
	Few developers had experience with even one alternative method with which to substitute for or extend XP practices. Experience of alternative agile methods (e.g., LSD, Crystal) was particularly lacking (3 of 16).	(7) Encourage developers to learn and gain experience of other methods (e.g., all 8 developers on P5's project team were tasked with learning and evaluating one agile method each (XP, XP Lite, Scrum, DSDM, Crystal, LSD, ASD, FDD).
	What experience did exist amongst the team was rarely elicited and used when deciding how to tailor/extend XP.	(9) Elicit developer knowledge and experiences of other methods and practices and incorporate into the tailoring/extension of XP.
Disciplined and purposeful approach to method tailoring	There was often little monitoring or control of adherence to XP practices following the initial implementation. In some cases this was beneficial as the actual users of the method decided how it should be used (e.g., P1). However, in some cases nonadherence was due to laziness or negligence and led to the gradual abandonment of the method (P3, P6, P12, P15).	(10) Frequently monitor adherence to XP practices, to ensure nonadherence is not simply due to laziness or negligence (e.g., at every retrospective meeting, P15's team reviewed the use of each practice, its pros and cons, and whether it should be retained).
	Conflict occasionally arose due to inconsistent adoption of practices across different members of the team (P3, P6, P12, P15).	(11) Communicate postimplementation tailoring efforts across the team (e.g., at standup or retrospectives). (12) If a tailoring decision is taken by an individual developer, its impact on the other team members should be assessed and discussed (e.g., at standup or retrospectives).

including the need to develop a framework to allow method comparison, and the identification of clusters of independent practices within methods, allowing developers to decompose a method into smaller fragments without fear of adverse consequences. This would pave the way for the development of more independent and decomposable agile methods. It also highlights an urgent need for researchers to learn more about what practices are interconnected and interdependent. The researcher participants in this study felt that, like so many other aspects of agile methods, very little is known as yet about such relationships and the implications of removing certain practices. Further opportunities for further research are listed in Table II.

In Table III we summarize our findings in relation to developer characteristics and propose a number of recommendations for software engineering practice which were found to aid tailoring in the projects studied.

This research focused solely on XP, interviewing both academics with active research experience of the method and expert practitioners with substantial hands-on exposure to it. While the recommendations for research and practice in Table II and III are oriented toward XP, there is no reason to think that these cannot be applied to software development methods in general. Future researchers could replicate this study, applying the newly derived framework to assess if the method tailoring problems and deficiencies identified in this study in relation to XP also arise with other methods.

REFERENCES

ABRAHAMSSON, P., SALO, O., RONKAINEN, J., AND WARSTA, J. 2002. Agile Software Development Methods: Review and Analysis. VTT Publications, Technical Research Centre of Finland, Espoo, Finland, 478.

ÅGERFALK, P. AND FITZGERALD, B. 2005. Methods as action knowledge: Exploring the concept of method rationale in method construction, tailoring and use. In *Proceedings of the 10th IFIP WG8.1 International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD)*.

ÅGERFALK, P. AND WISTRAND, K. 2003. Systems development method rationale: A conceptual framework for analysis. In *Proceedings of the 5th International Conference on Enterprise Information Systems (ICEIS)*.

AMBLER, S. 2007. Survey says. . .agile has crossed the chasm. *Dr. Dobb's J. World of Softw. Develop. 32*, 8, 59–61.

AMBLER, S. W. 2002. *Agile Modeling: Best Practices for the Unified Process and Extreme Programming*. John Wiley & Sons, New York, NY.

AUER, K. AND MILLER, R. 2002. *Extreme Programming Applied—Playing to Win*. Addison-Wesley, Reading, MA.

AVISON, D. AND FITZGERALD, G. 2003. *Information Systems Development: Methodologies, Techniques and Tools*. McGraw-Hill, London, U.K.

AVISON, D. AND WOOD-HARPER, A. 1991. *Multiview: An Exploration in Information Systems Development*. Blackwell Scientific Publications, Oxford, U.K.

AYDIN, M. AND HARMSEN, F., ET AL. 2004. An agile informations systems development method in use. *Turk. J. Electron. Eng. 12*, 2, 127–138.

BASKERVILLE, R. AND STAGE, J. 2001. Accommodating emergent work practices: Ethnographic choice of method fragments. In *Realigning Research and Practice in Is Development: The Social and Organizational Perspective*, B. FitzGerald, N. Russo, and J. DeGross, Eds. Kluwer, New York, NY, 12–28.

BASKERVILLE, R., AND TRAVIS, J., ET AL. 1992. Systems without method: The impact of new technologies on information systems development projects. In *The Impact of Computer Supported Technologies on Information Systems Development*, K. Kendall, J. DeGross, and K. Lyytinen, Eds. North Holland, Elsevier Science Publishers, Amsterdam, The Netherlands, 241–269.

BECK, K. 2000. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, Reading, MA.

BECK, K. AND ANDRES, C. 2004. *Extreme Programming Explained*, 2nd ed. Addison Wesley, Reading, MA.

BECK, K. AND BOEHM, B. 2003. Agility through discipline: A debate. *IEEE Comput. 36*, 6, 44–46.

BECK, K. AND FOWLER, M. 2001. *Planning eXtreme Programming*. Addison-Wesley, Boston, MA.

BENYON, D. AND SKIDMORE, S. 1987. Towards a toolkit for the systems analyst. *Comput. J. 30*, 1, 2–7.

BOEHM, B. AND TURNER, R. 2003. Using risk to balance agile and plan-driven methods. *IEEE Softw. 36*, 6, 57–66.

BOEHM, B. AND TURNER, R. 2004. *Balancing Agility and Discipline: A Guide For the Perplexed*. Addison-Wesley, Boston, MA.

BOOCH, G.   1994.   *Object-Oriented Analysis and Design with Applications*. Benjamin Gummings Publishers, Redwood City, CA.

BOWERS, J., MAY, J., ET AL.   2002.   *Tailoring XP for Large Mission Critical Software Development*. XP/Agile Universe, Chicago, IL.

BRINKKEMPER, S.   1996.   Method engineering: Engineering of information systems development methods and tools. *Inform. Softw. Tech. 38*, 4, 275–280.

BRINKKEMPER, S., SAEKI, M. ET AL.   1998.   Assembly techniques for method engineering. In *Advanced Information Systems Engineering*, G. Goos, J. Hartmanis, and J. van Leeuwen, Eds. Springer, Berlin, Germany, 381–390.

BRINKKEMPER, S., SAEKI, M., ET AL.   1999.   Meta-modelling based assembly techniques for situational method engineering. *Inform. Syst. 24*, 3, 209–228.

BROOKS, F.   1975.   *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, Reading, MA.

BROOKS, F.   1987.   No silver bullet: Essence and accidents of software engineering. *IEEE Comput. 20*, 4, 10–19.

BROWN, B.   1968.   *Delphi Process: A Methodology Used for the Elicitation of Opinions of Experts*. RAND Corporation, Santa Monica, CA.

CAO, L., MOHAN, K., ET AL.   2004.   How extreme does extreme programming have to be? Adapting XP practices to large-scale projects. In *Proceedings of the 37th Hawaii International Conference on System Sciences*. IEEE Computer Society Press, Los Alamitos, CA.

CLAYTON, M.   1997.   Delphi: A technique to harness expert opinion for critical decision-making tasks in education. *Edu. Psychol. 17*, 4, 373–387.

COAD, P. AND PALMER, S.   2002.   *Feature-Driven Development*. Prentice Hall, Englewood Cliffs, NJ.

COCKBURN, A.   2001.   *Crystal Clear: A Human-Powered Software Development Methodology for Small Teams*. Addison-Wesley, Reading, MA.

COCKBURN, A.   2002.   *Agile Software Development*. Addison-Wesley, Reading, MA.

COLEMAN, D., ARNOLD, P., ET AL.   1994.   *Object Oriented Development: The Fusion Method*. Prentice-Hall, Englewood Cliffs.

CONBOY, K.   2006.   A Framework of agility in information systems development. Ph.D dissertation. University of Limerick, Limerick, Ireland.

CONNORS, D. T.   1992.   Software development methodologies and traditional and modern information systems. *Softw. Eng. Notes 17*, 2, 43–49.

CRISPIN, L. AND HOUSE, T.   2003.   *Testing Extreme Programming*. Pearson, Boston, MA.

CRONHOLM, S. AND GOLDKUHL, G.   1994.   Meanings and motivates of method customization in CASE environments—observations and categorizations from an empirical study. In *Proceedings of the 5th Workshop on the Next Generation of CASE Tools*.

DALKEY, N. AND HELMER, O.   1963.   An experimental application of the Delphi method to the use of experts. *J. Institute Man. Sci. 9*, 3, 458–467.

DAVIS, G. B.   1982.   Strategies for information requirements determination. *IBM Syst. J. 21*, 1, 4–30.

DELBECQ, A. L., VAN DE VEN, A. H., ET AL.   1975.   *Group Techniques for Program Planning*. Scott Foresman and Company, Glenview IL.

DEMARCO, T.   1982.   *Controlling Software Projects: Management Measurement and Estimation*. Prenctice-Hall, Englewood Cliffs, NJ.

FITZGERALD, B.   1996.   Formalised systems development methodologies: A critical perspective. *Inform. Syst. J. 6*, 1, 3–23.

FITZGERALD, B., HARTNETT, G., ET AL.   2006.   Customising agile methods to software practices. *Eur. J. Inform. Syst. 15*, 2, 197–210.

FITZGERALD, B., RUSSO, N., ET AL.   2002.   *Information Systems Development: Methods in Action*. McGraw-Hill, New York, NY.

FOWLER, M. AND HIGHSMITH, J. 2001.   The agile manifesto. *Softw. Develop. 9*, 8, 28–32.

GLASS, R.   1991.   *Software Conflict: Essays on the Art and Science of Software Engineering*. Yourdon Press, Prentice Hall, Englewood Cliffs, NJ.

GREMILLION, L. AND PYBURN, P.   1983.   Breaking the systems development bottleneck. *Harvard Busin. Rev. 61*, 2, 130–137.

HARDY, C., THOMPSON, J., ET AL. 1995. The use, limitations and customization of structured systems development methods in the United Kingdom. *Inform. Softw. Tech. 37*, 9, 467–477.

HARMSEN, F. 1997. Situational method engineering. Ph.D. dissertation. Twente University, Enschede, Netherlands.

HARMSEN, F., BRINKKEMPER, S., ET AL. 1994. Situational method engineering for IS project approaches. In *Methods and Associated Tools for the IS Life Cycle*, A. Verrijn-Stuart and T. Olle, Eds. North-Holland, Elsevier Science, Amsterdam, The Netherlands, 169–194.

HENDERSON-SELLERS, B. AND SEROUR, M. 2005. Creating a dual-agility method: The value of method engineering. *J. Datab. Manag. 16*, 4, 1–23.

HIGHSMITH, J. 2004. *Agile Project Management*. Addison-Wesley, Reading, MA.

IIVARI, J. 1989. A methodology for IS development as organizational change. In *Syst. Development for Human Progress*, H. Klein and K. Kumar, Eds. North-Holland, Amsterdam, The Netherlands, 197–217.

JENKINS, A., NAUMANN, J., ET AL. 1984. Empirical investigation of systems development practices and results. *Inform. Man. 7*, 1, 73–82.

JOHNSON, D. AND CARISTI, J. 2003. eXtreme programming and the software design course. In *Extreme Programming Perspectives*, M. Marchesi, G. Succi, D. Wells, and L. Williams, Eds. Addison Wesley, Reading, MA, 47–59.

JOHNSON, J. 1995. Chaos: The dollar drain of IT project failures. *Appl. Develop. Trends 2*, 1, 41–47.

KARLSSON, F. AND AGERFALK, P. 2004. Method configuration: Adapting to situational characteristics while creating reusable assets. *Inform. Softw. Tech. 46*, 9, 619–633.

KEIL, M., MANN, J., ET AL. 2000. Why software projects escalate: An empirical analysis and test of four theoretical models. *MIS Quart. 24*, 4, 631–664.

KIRCHER, M., JAIN, P., ET AL. 2001. Distributed extreme programming. In *Proceedings of the Conference on XP2001—eXtreme Programming and Flexible Processes in Software Engineering*.

KIRCHER, M. AND LEVINE, D. 2001. The XP of Tao: eXtreme programming of large open-source frameworks. In *Extreme Programming Examined*. Addison-Wesley, Reading, MA, 463–485.

KOCH, A. 2005. *Agile Software Development: Evaluating the Methods for Your Organization*. Artech House, Norwood, MA.

KUMAR, K. AND WELKE, R. J. 1992. Methodology engineering: A proposal for situation-specific methodology construction. In *Challenges and Strategies for Research in Systems Development*, W. Cotterman and J. Senn, Eds. John Wiley & Sons Ltd., New York, NY, 257–269.

LEHMAN, M. 1978. Why software projects fail. In *Proceedings of the Infotech State of the Art Conference*. Pergamon Press, New York, NY.

LINBERG, K. 1999. Software developer perceptions about software project failure: A case study. *J. Syst. Softw. 49*, 1, 177–192.

LINSTONE, H. AND TUROFF, M. 1975. Introduction. *The Delphi Method: Techniques and Applications*, H. Linstone and M. Turoff, Eds., Addison-Wesley, Reading, MA, 3–12.

MARTIN, R. 2003. *Agile Software Development: Principles, Patterns and Practices*. Prentice-Hall, Upper Saddle River, NJ.

MCBREEN, P. 2003. *Questioning Extreme Programming*. Addison-Wesley, Reading, MA.

MCCRACKEN, G. 1988. *Qualitative Research: The Long Interview*. Sage Publications, London, UK.

MCDOWELL, C., WERENER, L., ET AL. 2003. The impact of pair programming on student performance, perception and persistence. In *Proceedings of the 25th International Conference on Software Engineering*. Portland, OR.

MEYER, M. AND BOOKER, J. 2001. *Eliciting and Analyzing Expert Judgment: A Practical Guide*. Society for Industrial Mathematics, Boston, MA.

MILES, M. AND HUBERMAN, A. 1999. *Qualitative Data Analysis*. Sage, London, U.K.

MIRBEL, I. AND RALYTE, J. 2006. Situational method engineering: Combining assemble-based and roadmap-driven approaches. *J. Requirement Engin. 11*, 1, 58–78.

MOORE, C. 1987. *Group Techniques for Idea Building*. Sage Publications, London, U.K.

NAUMANN, J., DAVIS, G., ET AL. 1980. Determining information requirements: A contingency method for selection of a requirements assurance strategy. *J. Syst. Softw. 1*, 4, 273–281.

NECCO, C., GORDON, C., ET AL. 1987. Systems analysis and design: Current practices. *MIS Quart. 11*, 3, 461–476.

OFFENBEEK, M. AND VAN KOOPMAN, P. 1996. Scenarios for systems development: matching context and strategy. *Behav. Inform. Tech. 15*, 4, 250–265.

OPPENHEIM, A. 1992. *Questionnaire Design, Interviewing and Attitude Measurement*. Continuum, New York, NY.

POOLE, C. AND HUISMAN, J. 2001. Using extreme programming in a maintenance environment. *IEEE Softw. 18*, 6, 42–50.

POPPENDIECK, M. 2001. Lean programming. *Softw. Develop. Mag. 9*, 5, 71–75.

RASMUSSON, J. 2003. Introducing XP into greenfield projects: Lessons learned. *IEEE Comput. 20*, 3, 21–28.

ROBEY, D. AND KEIL, M. 2001. Blowing the whistle on troubled software projects. *Comm. ACM 44*, 4, 87–93.

ROLLAND, C. AND PRAKASH, N. 1996. A proposal for context-specific method engineering. In *Method Engineering: Principles of Method Construction and Tool Support*, S. Brinkkemper, K. Lyytinen, and R. Welke, Eds. Chapman & Hall, Atlanta, CA, 191–208.

RUSSO, N., WYNEKOOP, J., ET AL. 1995. The use and adaptation of system development methodologies. In *Managing Information & Communications in a Changing Global Environment*, M. Khosrowpour, Eds. Idea Group Publishing, Philadelphia, PA, 843–844.

SACKMAN, H. 1975. *Delphi Critique*. Heath and Co., Lexington, MA.

SCHWABER, C. AND FICHERA, R. 2005. Corporate IT leads the second wave of agile adoption. Forrester res. rep. Forrester Research, Cambridgr, MA.

SCHWABER, K. 1996. Controlled chaos: Living on the edge. *Amer. Program. 9*, 5, 10–16.

SCHWABER, K. AND BEEDLE, M. 2002. *Agile Software Development with Scrum.* Prentice-Hall, Upper Saddle River, NJ.

SOMMERVILLE, I., AND RANSOM, J. B. 2005. An industrial experiment in requirements engineering process assessment and improvement. *ACM Trans. Softw. Eng. Methodol. 14*, 1–33.

STAPLETON, J. 1997. *DSDM: Dynamic Systems Development Method.* Addison Wesley, Reading, MA.

STEPHENS, M. AND ROSENBERG, D. 2003. *Extreme Programming Refactored*. Apress, New York, NY.

STOTTS, D., WILLIAMS, L., ET AL. 2003. *Virtual Teaming: Experiments and Experiences with Distributed Pair Programming*. Extreme Programming/Agile Universe. Springer, New Orleans, L.A.

SULLIVAN, C. H. 1985. Systems planning in the information age. *Sloan Busi. Rev. 26*, 2, 3–11.

TAN, C. AND TEO, H. 2007. Training future software developers to acquire agile development skills. *Comm. ACM 50*, 12, 97–98.

TOLVANEN, J. AND LYYTINEN, K. 1993. Flexible method adaptation in CASE—the metamodeling approach. *Scand. J. Inform. Syst. 5*, 1, 51–77.

TRAUTH, E. AND O'CONNOR, B. 1991. A study of the interaction between information, technology and society. *Information Systems Research: Contemporary Approaches and Emergent Traditions*, H. Nissen, H. Klein, and R. Hirschheim, Eds., North Holland, Elsevier, Amsterdam, The Netherlands, 131–144.

TRUEX, D., BASKERVILLE, R., ET AL. 1999. Growing systems in emergent organizations. *Comm. ACM 42*, 8, 117–123.

UHL, N. 1983. *Using Research for Strategic Planning.* Jossey-Bass, San Francisco, CA.

VIJAYASARATHY, L. AND TURK, D. 2008. Agile software development: A survey of early adopters. *J. Inform. Tech. Man. 19*, 2, 1–8.

WAINER, M. 2003. *Adaptations for Teaching Software Development with eXtreme Programming: An Experience Report*. XP/Agile Universe, Springer.

WOOD-HARPER, A., ANTILL, L., ET AL. 1985. *Information Systems Definition: The MultiView Approach*. Blackwell Publishers, Oxford, U.K.

WYNEKOOP, J. L. AND RUSSO, N. L. 1995. Systems development methodologies: Unanswered questions. *J. Inform. Tech. 10*, 2, 65–73.

YIN, R. 2003. *Case Study Research: Design and Methods.* Sage Publications, Thousand Oaks, CA.