Reconciling software development models: A quasi-systematic review

Andréa Magalhães Magdaleno¹,²,*, Cláudia Maria Lima Werner³, Renata Mendes de Araujo¹,²,³

¹ COPPE/UFRJ – Systems Engineering and Computer Science Program, Federal University of Rio de Janeiro (UFRJ), P.O. Box 68511, Rio de Janeiro 21945-970, RJ, Brazil
² NP2TeC – Research and Practice Group in Information Technology, Brazil
³ Graduate Program in Informatics (PPGI), Federal University of Rio de Janeiro State (UNIRIO), Urca, Rio de Janeiro, RJ, Brazil

A R T I C L E   I N F O

Article history:
Received 24 January 2010
Received in revised form 22 August 2011
Accepted 23 August 2011
Available online xxx

Keywords:
Systematic review
Software process
Reconciliation among development models
Plan-driven
Agile
Free/open source software

A B S T R A C T

Purpose: The purpose of this paper is to characterize reconciliation among the plan-driven, agile, and free/open source software models of software development.

Design/methodology/approach: An automated quasi-systematic review identified 42 papers, which were then analyzed.

Findings: The main findings are: there exist distinct – organization, group and process – levels of reconciliation; few studies deal with reconciliation among the three models of development; a significant amount of work addresses reconciliation between plan-driven and agile development; several large organizations (such as Microsoft, Motorola, and Philips) are interested in trying to combine these models; and reconciliation among software development models is still an open issue, since it is an emerging area and research on most proposals is at an early stage.

Research limitations: Automated searches may not capture relevant papers in publications that are not indexed. Other data sources not amenable to execution of the protocol were not used. Data extraction was performed by only one researcher, which may increase the risk of threats to internal validity.

Implications: This characterization is important for practitioners wanting to be current with the state of research. This review will also assist the scientific community working with software development processes to build a common understanding of the challenges that must be faced, and to identify areas where research is lacking. Finally, the results will be useful to software industry that is calling for solutions in this area.

Originality/value: There is no other systematic review on this subject, and reconciliation among software development models is an emerging area. This study helps to identify and consolidate the work done so far and to guide future research. The conclusions are an important step towards expanding the body of knowledge in the field.

© 2011 Elsevier Inc. All rights reserved.

1. Introduction

Software development organizations are continually challenged by the need to improve the quality of software products. Technologies are changing rapidly and software is becoming larger and more complex. In addition, large-scale, distributed development poses new challenges.

In this context, the assumption that the software development process directly influences the quality of the product developed (Cugola and Ghezzi, 1998; Pressman, 2001; Raman, 2000) has motivated many organizations to adopt maturity models, such as CMMI-DEV (Capability Maturity Model Integration) (Chrisisis et al., 2006), or quality standards, like ISO 12207 (ISO/IEC, 1995). As a result, the plan-driven development model has become less chaotic, more predictable and more manageable.

The success of some free/open source software (FOSS) projects, such as Linux, Apache and Mozilla, also caught the attention of academia, industry and users, because of their capability to produce high-quality software quickly and for free (Cubranic and Booth, 1999; Feller and Fitzgerald, 2001; Raymond, 2001). The FOSS development model does not have a formal, disciplined definition, since it works based predominantly on voluntary collaboration.

Agile methods, using shorter development cycles and higher levels of client involvement and participation to cope with the changes that occur in the course of a development project, are often presented as an alternative to plan-driven software development (Dyba and Dingsoyr, 2008). Agile methods advocate
software development with minimal documentation, based on self-organized teams that use the tacit knowledge acquired during the project (Cockburn, 2001).

Each of these three development models has its peculiarities, successes and challenges, and has taken separate paths. Due to differences in vocabulary, misinterpretation and misuse of approaches, a common view is that plan-driven and agile are poles apart (Boehm, 2002; Boehm and Turner, 2003b; Glass, 2001; Glazer et al., 2008; Santana et al., 2009). While plan-driven and FOSS also tend to be perceived as opponents, FOSS and agile tend to be perceived as more similar than different (Abrahamsson et al., 2002; Simmons and Dillon, 2003; Theunissen et al., 2008; Warsta and Abrahamsson, 2003). Both agile and FOSS emphasize iterative development and frequent releases, with minimum documentation and the focus on testing (Barnett, 2004). However, it is also important to note that there is a key difference between FOSS and agile: several agile methods (like Extreme Programming) emphasize collaboration, while FOSS typically involves geographically distributed developers. Another contrast is that FOSS relies on tools to make its distributed approach work, while agile methods do not. Therefore, FOSS is similar to plan-driven in all the same respects that it differs from agile: both FOSS and plan-driven favor distributed development, as well as the use of tools to support development tasks.

In order to examine how the particularities of each development model might be accommodated, we conducted a quasi-systematic review of the technical literature with a view to characterizing reconciliation among software development models. Reconciliation is defined as the act of re-establishing normal relations between belligerents. In software development, it is an attempt to approximate the different development models in order to establish more effective software processes. Since there is no other systematic review on this topic and reconciliation is an emerging area, this study is useful to identify and consolidate work to date and to guide future research. The conclusions are an important step towards expanding the body of knowledge in the area.

The remainder of this paper is organized as follows. In Section 2, we present the main characteristics of each software development model. Section 3 describes the main objectives and phases of the systematic review process. Section 4 details the plan of this quasi-systematic review, defining a review protocol that specifies the main and secondary research questions. In Section 5, the execution of the protocol outlined in Section 4 is described. In Section 6, we report and analyze the results of this quasi-systematic review. In Section 7, the research questions are discussed and answered, characterizing reconciliation among software development models. Finally, Section 8 concludes the paper, discusses its limitations and indicates some opportunities for future work.

2. Software development models

A software development model defines, at a high-level of abstraction, a set of best practices for software development (Sommerville, 2004). These practices are organized into a software process that corresponds to the “set of Software Engineering tasks needed to transform user requirements into software” (Humphrey, 1989). Fuggetta (2000) defined a software process as “a coherent set of policies, organizational structures, technologies, procedures, and artifacts required to design, develop, deploy and maintain a software product”.

Plan-driven, agile and FOSS development models have the same goal: to improve the software development process, but they adopt different approaches. In the past decade, these three models have had an enormous impact on software industry and their future outlooks are also promising (Ebert, 2007; Theunissen et al., 2008).

Each model represents a distinct development universe, differing in philosophy and main characteristics. While the plan-driven model seeks predictability, stability and reliability (Chrisiss et al., 2006), agile development tries quickly to add value to business and adapt to changes (Cockburn, 2001). Meanwhile, in FOSS development, the main objective is to guarantee user freedom (FSF, 2008).

The plan-driven model is typically exemplified by maturity models such as CMMI-DEV (Chrisiss et al., 2006), or quality standards, like ISO 12207 (ISO/IEC, 1995). Historically, this development model was designed to handle high-risk environments and large and complex projects with high costs involved, where relationships between client and development team were characterized by low trust and were governed by contractual definitions (Glazer et al., 2008). Besides, due to this lack of trust, frequent approvals are necessary in each stage of the development lifecycle. There is also a concern with the completeness of documentation and the traceability between documentation generated in each phase. In this scenario, the plan-driven model is characterized by its orientation to detailed planning and emphasis on well-defined and continuous improved processes.

The assumptions of agile development, as observed in methods such as Extreme Programming (XP) (Beck, 2004), Scrum (Schwaber, 2004), Crystal (Cockburn, 2004), Kanban (Anderson, 2010) can be summarized by the four values of the Agile Manifesto (Beck et al., 2001): individuals and interactions over processes and tools; software running over comprehensive documentation; customer collaboration over contract negotiation; and responding to change over following a plan. In general, the main objectives of agile methods are to deliver value quickly to customers and to be responsive to changes in market, technology and environment (Cockburn, 2001). On this approach to adaptation, the project is implemented incrementally, with short development cycles (Miller, 2001) and a co-located team.

Free/open source software offers users four basic and irrevocable freedoms: to run, study, improve and distribute the program code (FSF, 2008). Access to program source code is a prerequisite for these freedoms. The FOSS development model does not comprise any technological breakthrough, but rather represents a new manner of developing software products based on special characteristics. Therefore, FOSS is also considered a development model with specific practices for the main phases of a development process (DiBona et al., 1999). It can be understood by the metaphor of the bazaar (Raymond, 2001), where the projects are developed collaboratively and transparently. Users are welcome to participate and provide feedback to the development team at any stage. Developers with different levels of skill and availability, geographically distributed around the world, work on a voluntary basis, using the Internet as a communication channel (Capiuluppi et al., 2003).

From the main characteristics of the software development models presented, it can be seen that they have both similarities and significant differences. For instance, agile is more iterative and incremental than plan-driven, and focuses on customer collaboration, while plan-driven focuses more on following processes and explicitly documenting as much as possible. FOSS development is close to agile methods in terms of iterative and incremental development, but differs in its distributed (as opposed to co-located) development. In addition, FOSS needs tools to make its distributed approach work, while agile methods do not have this focus. Therefore, in this respect, FOSS tends to be more similar to plan-driven and less similar to agile methods.

Therefore, each one addresses better, or works better on, different characteristic issues in software development projects. Since any given development project may require practices from different models in order to be effective, research in the area has discussed how to accommodate the particularities of each model in development processes, so as to achieve better productivity.
3. Systematic review process

A systematic literature review (Biollchini et al., 2005; Brereton et al., 2007; Kitchenham, 2004) is a type of secondary study, i.e., a study based on previously published research. It is a research methodology developed to gather, evaluate, and analyze all the available literature relevant to a particular research question, topic area, or phenomenon of interest (Kitchenham et al., 2007; Kitchenham and Charters, 2007).

The main reason for performing a systematic review is to improve the quality of the material covered in the subject of interest, as compared with what is obtained by a less formal review. By preventing unnecessary duplication of effort and error it can help in planning the research, and can orient the investigation process.

In contrast with a conventional literature review conducted in an ad-hoc manner at the start of a study, a systematic review follows a strict, well-defined sequence of methodological steps (Fig. 1), which yield high scientific value and reliable results. A systematic review requires considerable effort in comparison to a conventional literature review, but offers complete, in-depth research in an area of interest (Kitchenham and Charters, 2007), ensuring important papers are not overlooked or incorrect conclusions drawn (MacDonell et al., 2010).

The concept of systematic review emerged in the field of medicine, and its adaptation to Software Engineering is presented in Kitchenham (2004). Systematic Literature Reviews have been gaining significant attention from software engineering researchers since 2004. Several researchers (Dyba and Dingsøyr, 2008; Kitchenham et al., 2009a,b; 2007; MacDonell et al., 2010; Pedreira et al., 2007) have reported their experiences and lessons learned from applying systematic reviews to different subject matter in Software Engineering (Babar and Zhang, 2009). The method proposed consists of three main phases: planning, execution, and reporting. Fig. 1 gives an overview of the systematic review process used in this study. Each of the activities is described in the sections below.

During the planning phase, the researcher identifies the review objectives, specifies the research question, and develops the review protocol (see Section 4). At the end of this phase, the protocol must be validated before being applied, in order to establish the feasibility of the plan.

In the execution phase, searches are performed in the defined publication sources. The studies returned are evaluated according to the criteria established in the protocol. The relevant data found in the selected papers are extracted and synthesized.

In the third phase, the results of the review are reported. In practice, this phase is performed throughout the whole process, storing results from all previous phases. The whole process, including intermediate results, must be documented.

In this study, although the survey maintains the same formal characteristics of a systematic review as described above, it is considered a quasi-systematic review (Travassos et al., 2008). This is because it is an exploratory study designed to characterize the research area, and no baseline is established for comparison of the results obtained.

4. Planning phase

This quasi-systematic review begins by defining a review protocol that specifies the main and secondary research questions. The protocol should specify the inclusion and exclusion criteria and document the search strategy used in order to allow readers to ascertain the degree of accuracy and completeness.

The following subsections describe the components of the protocol used in this review. This protocol was defined based on the template proposed by Biollchini et al. (2005). The protocol was created by the authors and reviewed by an independent researcher who is a specialist in Empirical Software Engineering, including systematic reviews.

4.1. Objectives and question formulation

The objective of this review is to characterize reconciliation among plan-driven, agile, and FOSS development models. However, it was not intended to investigate the individual characteristics of each software development model or report on the use of individual approaches in software development projects. To achieve this goal, this quasi-systematic review aims to answer the following main question (MQ) and secondary questions (SQ):

MQ: How can organizations reconcile among plan-driven, agile, and FOSS software development models?

SQ1: What are the opportunities and challenges in reconciliation?

SQ2: What proposals have been made for reconciliation?

SQ3: What reconciliation strategies have been adopted by organizations or communities?

SQ4: What results have been achieved by organizations or communities attempting reconciliation?

4.2. Review scope

The scope of this review was defined according to the PICO (Pai et al., 2004) approach that structures the research question into four basic elements: Population, Intervention, Comparison and Outcome.

- Population: software development organizations and software development projects.
- Intervention: software development process (plan-driven, agile, and free/open).
- Comparison: Not applied.
Outcomes:
- Characterization of reconciliation among software development models.
- Opportunities and challenges for reconciliation among software development models.
- Proposals for reconciliation among software development models.
- Strategies for reconciliation among software development models.
- Results obtained with reconciliation among software development models.

4.3. Control papers

The control papers were obtained during a previous conventional literature review. Control papers are helpful to provide an initial understanding of the area in order to define search keywords (Section 4.5). They are also used to test and calibrate the searches in the electronic databases. The three papers used as control in this search were:


4.4. Data sources and search strategy

The search strategy included the following electronic databases: Compendex, IEEE Xplore, Scopus, and Web of Science. These libraries were chosen because they permit article full text recovery. They also permit automated searches of article content. Although broad automated searches take more time than narrow manual searches, they also return larger numbers of relevant studies (Kitchenham et al., 2009a). Therefore, these sources were considered significant in order to offer relevant content and contribute to the search result.

The ACM Library, despite its importance in the field of Computer Science, overlaps with the IEEE Xplore library and its content is also indexed by the Scopus library. Thus, although this library was not included among the selected sources, we consider that its content was covered.

4.5. Keywords

This section describes the main terms that composed each research question. The keywords were constructed considering (Kitchenham et al., 2007): terms in population, intervention, and outcomes (Section 4.2); alternative spellings and synonyms for these terms; and keywords identified in the control papers (Section 4.3).

The complete list of keywords used in this quasi-systematic review is given in Table 1, where population and intervention can be seen to be the same in the main question (MQ) and in every secondary question (SQ), since these are a subset of the main question.

4.6. Inclusion and exclusion criteria

This review includes every article returned by the protocol that meets at least one of the following criteria for inclusion (IC) and does not meet any of the criteria for exclusion (EC):

- IC1—Documents must address reconciliation between two or more software development models;
- IC2—Documents must discuss opportunities and challenges for reconciliation among software development models;
- IC3—Documents must present proposals for reconciliation among software development models;
- IC4—Documents must report experiences of organizations or communities that have implemented practices from more than one software development model.

Publications that satisfy at least one of the following EC were excluded:

- EC1—Documents not written in English;
- EC2—Documents whose full text is not available;
- EC3—Documents clearly dealing with topics irrelevant to the purpose of this review;
- EC4—Documents merely reporting the use of individual software processes in development projects;
- EC5—Documents not addressing the software development models, but tools and computing environments to support them;
- EC6—Documents addressing reconciliation, but focusing on legal aspects, business model or evaluation methods, and not on the development process itself;
- EC7—If the same study has been published more than once, the most relevant version (i.e., the one explaining the study in greatest detail) will be used and the others will be excluded;
- EC8—If a given study has been selected for a broader research question, it must be excluded from the list of selections for the narrower research question.

4.7. Selection process

Selection of articles occurred in four steps:

1. Selection and preliminary organization of collected documents: a preliminary selection of publications was made by applying the search string (Section 4.10) to selected data sources (Section 4.4). Each publication was cataloged to a reference repository for later analysis.
2. Selection of relevant papers: primary selection using the search string cannot guarantee that all material collected is useful in the research context, because application of search expressions is restricted to syntactic considerations. Thus, after the identification of publications via search engines, the documents were reviewed in view of the inclusion and exclusion criteria. One of the authors was responsible for this step.
3. Evaluation of relevant papers: the other two authors and three students evaluated the list of documents included and excluded. Where assessments conflicted, the researchers had to reach agreement on selection of publications. To reduce the risk of excluding a publication at an early stage of the study, when the researchers were in doubt the publication was included.
4. Information extraction from relevant documents: after defining the final list of relevant documents, one of the authors read them to extract information on reconciliation among software development models. This step was also evaluated by sampling under the responsibility of the other two researchers.

4.8. Assessment of the quality of work

In this study, it is assumed that the document sources are reliable, and that the texts have undergone external reviews that serve as a filter assuring that they have a sufficient quality to contribute to this quasi-systematic review.

In the event of conflicting information among the documents, greater prominence will gradually be given to articles that: put forward the idea of reconciliation among models; offer a theoretical proposal for the reconciliation; and report the results of actual experiences of organizations that adopted reconciliation among software development models.

4.9. Information extraction

The data extraction strategy is designed to gather the information necessary to answer the research questions and evaluate the quality of the paper. Thus, for each document approved by the selection process, one of the authors extracts, and the others review by sampling, the information defined in the extraction form (see Appendix A). This form enabled us to record full details of the papers under review and to be specific about how each of them addressed our research questions.

4.10. Search string

According to Pai et al. (2004) the four basic elements – population (P), intervention (I), comparison (C), and outcome (O) – that structure the research question can be related by the AND logical operator. Since comparison (C) is not applicable to this study, this was done for the set of keywords chosen to represent population (P), intervention (I) and outcome (O), resulting in the following structure: (P) AND (I) AND (O). Within each of these three elements of the structure, their keywords were combined with the OR operator (Table 2).

Due to the large number of terms, it was decided to divide the search into multiple strings, according to possible combinations of development models. This reduced search string complexity, ensuring better readability and decreasing the probability of human and search engine error. In this case, each research question was split into four parts, as shown in the Models column of Table 2: (a) combination of the three development models; (b) combination of agile and free/open source; (c) combination of plan-driven and free/open source; and (d) combination of plan-driven and agile.

The search strings were derived from the research questions (Section 4.1) and the keywords (Section 4.5). The strings used for the main question are listed in Table 2. Since MQa, MQb, MQc, and MQd vary only in the development model in intervention, all of them have the same population and outcome. The same procedure was used in the secondary questions.

5. Execution phase

After the planning phase, the review protocol can be applied. This section describes in detail how this execution phase was conducted.

5.1. Search execution

The first round of searches in each of the selected digital libraries was performed in April 2009. In the Compendex library, the search was performed with the autostemming option and using the summary field in the search. Due to the small number of documents returned by the original string, only the population and intervention were considered to search, ignoring the result, and absorbing the effort of manually selecting documents. As the Compendex machine returned duplicate items to the same string, these duplicates were promptly excluded. Only two of the three control papers were indexed by Compendex library, and were successfully retrieved by the search.

The IEEE Xplore library returned few documents with the original string, which had to be reworded to increase its range. Thus, it was also considered to search only the population and intervention. To improve the quality of the results, the search was performed using the summary field. Of the two control papers indexed by the library, only one was recovered.

The Scopus database was searched using the title, abstract and keyword fields. The three control papers were indexed and recovered.

Lastly, the Web of Science search engine failed to process the full string. Due to implementation limitations, this machine is restricted to 50 search terms and 49 operators. Thus, the string used did not include the outcome, but was limited to population and intervention. Also, the topic (TS) field was chosen for the search, since it includes title, abstract and keyword fields. It was noted, using the control papers, that the number of results increased if the terms were included also in plural form. Thus, the original strings (in Section 4.10) were changed to include a simplification of the population terms, and the plural terms, while at the same time keeping to a limited number of terms.

To summarize, Table 3 shows the consolidation of the original number of papers returned by searches performed for the main questions in each digital library. This total represents the raw data and is useful in order to understand how many papers were filtered in the following steps.

5.2. Elimination of duplicates

As a first step in the analysis of retrieved documents, the repetitions found in searches were eliminated, keeping the remaining paper accounted for the digital library with the largest number of items.
Table 2: Search strings.

<table>
<thead>
<tr>
<th>Question</th>
<th>Models</th>
<th>Population</th>
<th>Intervention</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQa</td>
<td>Plan-driven, agile, and FOSS</td>
<td>(“software development” OR “software engineering” OR “software project” OR “system development” OR “system engineering” OR “system project” OR “application development” OR “application engineering” OR “application project” OR “product development” OR “product engineering” OR “product project” OR “software organization” OR “software organisation” OR “software enterprise” OR “software corporation” OR “software company” OR “software industry” OR “software firm” OR “software community”)</td>
<td>AND (&quot;open source&quot; OR &quot;free source&quot; OR libre OR OSS OR FOSS OR FLOSS) AND ( agile OR lightweight OR lean OR XP OR Scrum) AND ( conventional OR traditional OR &quot;plan-driven&quot; OR &quot;plan driven&quot; OR closed OR rigorous OR proprietary OR CMM OR CMMI) AND ( process OR activity OR method OR approach OR practice OR methodology OR technique OR paradigm OR procedure OR principle)</td>
<td></td>
</tr>
<tr>
<td>MQb</td>
<td>Agile and FOSS</td>
<td></td>
<td>AND (&quot;open source&quot; OR &quot;free source&quot; OR libre OR OSS OR FOSS OR FLOSS) AND ( agile OR lightweight OR lean OR XP OR Scrum) AND ( process OR activity OR method OR approach OR practice OR methodology OR technique OR paradigm OR procedure OR principle)</td>
<td></td>
</tr>
<tr>
<td>MQc</td>
<td>Plan-driven and FOSS</td>
<td></td>
<td>AND (&quot;open source&quot; OR &quot;free source&quot; OR libre OR OSS OR FOSS OR FLOSS) AND ( conventional OR traditional OR &quot;plan-driven&quot; OR &quot;plan driven&quot; OR closed OR rigorous OR proprietary OR CMM OR CMMI) AND ( process OR activity OR method OR approach OR practice OR methodology OR technique OR paradigm OR procedure OR principle)</td>
<td></td>
</tr>
<tr>
<td>MQd</td>
<td>Plan-driven and agile</td>
<td></td>
<td>AND (agile OR lightweight OR lean OR XP OR Scrum) AND ( conventional OR traditional OR &quot;plan-driven&quot; OR &quot;plan driven&quot; OR closed OR rigorous OR proprietary OR CMM OR CMMI) AND ( process OR activity OR method OR approach OR practice OR methodology OR technique OR paradigm OR procedure OR principle)</td>
<td></td>
</tr>
</tbody>
</table>

From the total of 1078 publications examined, 260 (24% of the total) were duplicates, and excluded. Thus, 818 papers were selected for analysis in the following stage. This percentage of repetitions is a reflection of the overlapping among documents indexed in different digital libraries. In particular, the IEEE Xplore database was observed to overlap in many cases with the Scopus and Compendex libraries. However, to ensure greater coverage, all search engines were maintained, as some papers were indexed by only one of them.

After excluding duplicates, we calculated the percentage distribution of publications found by the search engines. From Fig. 2 it can be seen that results from the Compendex library predominated.

5.3. First filter: title and abstract

After elimination of duplicates, a first assessment was made by reading all the titles and abstracts and applying the exclusion criteria. Papers with titles and abstracts that clearly indicated they were outside the scope of this quasi-systematic review were excluded. As an example, since our search strategy included the term “xp and software”, we got several hits on Microsoft’s Windows XP operating system.

In addition, titles and abstracts are not always clear indicators of what a paper is about. If it was unclear whether a study matched the screening criteria, it was included for detailed quality assessment in the next stage.

Table 3: Overall results of the main question searches.

<table>
<thead>
<tr>
<th>Library</th>
<th>MQa</th>
<th>MQb</th>
<th>MQc</th>
<th>MQd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compendex</td>
<td>17</td>
<td>55</td>
<td>171</td>
<td>193</td>
</tr>
<tr>
<td>IEEE Xplore</td>
<td>4</td>
<td>10</td>
<td>36</td>
<td>100</td>
</tr>
<tr>
<td>Scopus</td>
<td>14</td>
<td>56</td>
<td>19</td>
<td>73</td>
</tr>
<tr>
<td>Web of Science</td>
<td>4</td>
<td>30</td>
<td>184</td>
<td>112</td>
</tr>
<tr>
<td>Total</td>
<td>39</td>
<td>151</td>
<td>410</td>
<td>478</td>
</tr>
</tbody>
</table>

From the total of 818 papers analyzed at this stage, 610 were excluded and 208 items were selected for analysis in the next phase, representing 25% of the studies analyzed in this phase (Table 4).

After application of the first filter by one author, the list of excluded items was revised by five other researchers (the other two authors and three students). The purpose of this validation is to ensure that items are not excluded improperly, and that relevant work is not lost. So, in the event of doubt or deadlock between
Table 4

Results of searches after the first filter.

<table>
<thead>
<tr>
<th>Library</th>
<th>MQa</th>
<th>MQb</th>
<th>MQc</th>
<th>MQd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compendex</td>
<td>6</td>
<td>2</td>
<td>42</td>
<td>63</td>
</tr>
<tr>
<td>IEEE Xplore</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>22</td>
</tr>
<tr>
<td>Scopus</td>
<td>2</td>
<td>16</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Web of science</td>
<td>0</td>
<td>1</td>
<td>37</td>
<td>5</td>
</tr>
<tr>
<td>Total</td>
<td>8</td>
<td>20</td>
<td>86</td>
<td>94</td>
</tr>
</tbody>
</table>

Table 5

Results of searches of the main question after the second filter.

<table>
<thead>
<tr>
<th>Library</th>
<th>MQa</th>
<th>MQb</th>
<th>MQc</th>
<th>MQd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compendex</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>17</td>
</tr>
<tr>
<td>IEEE Xplore</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Scopus</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Web of science</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>Total</td>
<td>1</td>
<td>4</td>
<td>10</td>
<td>27</td>
</tr>
</tbody>
</table>

Table 6

Results of secondary question searches after the second filter.

<table>
<thead>
<tr>
<th>Models</th>
<th>SQ1</th>
<th>SQ2</th>
<th>SQ3</th>
<th>SQ4</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) Plan-driven, agile, and FOSS</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(b) FOSS and agile</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(c) FOSS and plan-driven</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(d) Plan-driven and agile</td>
<td>9</td>
<td>18</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Total</td>
<td>14</td>
<td>27</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

researchers, the publication was included in the list of selected papers. The results shown below were consolidated after this validation.

5.4. Second filter: paper recovery and reading

Accurate, detailed assessment requires reading the full text of papers. So we started recovering the files with the complete paper texts. However, it was not possible to access all publications, even after contacting the authors by e-mail. Of the 208 items that should have been assessed at this stage, full texts of 182 (88%) were recovered through search engines or personal e-mail contact with the authors. Accordingly, it was not possible to access the full text of 26 papers.

The papers retrieved were read in order to select candidates for systematic review (Table 5). All papers were analyzed according to the inclusion and exclusion criteria. The complete list of the 42 papers selected is detailed in Appendix B. As regards the secondary questions, the results of applying the second filter are summarized in Table 6, where the same paper may address more than one SQ.

5.5. Information extraction

During this stage, all the information necessary to answer the research questions was extracted from the 42 studies selected in this review. The information was captured from each selected paper, as in the form shown in Appendix A. Extracted data were held in tables, one file per paper. All the resulting forms were grouped by the research questions they address.

5.6. Instruments

All references retrieved were imported into JabRef, the reference manager used as the main tool in this quasi-systematic review. The tool was useful to identify duplicates. In addition, it permitted custom fields to be created to register the extracted information, to maintain the track of the source database and to control the decision at each phase of the filtering process.

JabRef was also used to perform secondary question searches within the main question results. It also offers the possibility to create specific reports to export all the information in a predefined format to facilitate the work of handling references and information analysis.

6. Reporting phase

This section reports and discusses the results obtained by the quasi-systematic review. Initially, execution of the protocol in the search engines returned a total of 1078 publications. These papers were submitted to a filtering process, comprising four steps, as described in the sections above: elimination of duplicates, filtering by title and abstract, filtering by availability of full-text, and filtering by reading the full text of the paper. Fig. 3 shows the number of items returned from each digital library during each step of the filtering process.

Finally, 42 papers were actually selected as the result of the quasi-systematic review execution. This means that only 3% of the documents initially obtained actually contributed to the quasi-systematic review. The vast majority of papers read were excluded for lack of any direct contribution to characterizing reconciliation among software development models.

Of the exclusion criteria that were applied, the main reasons for excluding studies were: focus on individual software development processes only (EC4), lack of relevance to the study theme (EC3), and focus on tools and computing environments rather than software development models (EC5).

The citation among the 42 papers obtained during the quasi-systematic review, is shown in Fig 4. This graphical representation helps to identify the most influential publications. The most-cited papers are highlighted, showing clearly the influence of Boehm and Turner’s papers on this research topic. Although no time restriction was placed on the search, the results date from only 2002, showing the lack of maturity of this research field and the need, underscored by some authors, for more scientific papers on the topic. The field is also recent because of the initial perception that the plan-driven, agile, and FOSS models were opponents, and that there was no room for any attempt to integrate them.

6.1. Threats to validity

For a complete analysis of the results obtained with this quasi-systematic review, it is also important to consider the limitations of this review.

In this regard, the main threat to validity identified is that other relevant works are not indexed by the main search engines, due to the lesser standing of the publication. For instance, some papers (Glass, 2001; Glazer et al., 2008; Warsta and Abrahamsson, 2003) were identified in the conventional literature review and used as basis for preparing the research protocol, although they could not be used as control papers to calibrate the searches in search engines.

Regarding internal threats to validity, it is important to consider that data extraction was performed by only one researcher, which may entail some risk of bias. Although three students were involved in the paper evaluation filter, they did not participate in the data extraction step, which the other authors reviewed by sampling.

Finally, as mentioned in Section 5.4, another threat to validity is that it was not possible to access the full text of 26 papers, despite efforts to contact authors by e-mail. These represent approximately 2% of all publications retrieved, and some of them, after the last
7. Reconciling software development models

The purpose of this section is to characterize reconciliation among software development models, according to the results of the quasi-systematic review. Since the research question was further divided into a set of sub-questions, each of these questions is discussed and analyzed individually. Finally, some general conclusions are drawn.

7.1. MQ: how can organizations reconcile plan-driven, agile, and FOSS software development models?

This quasi-systematic review yielded a number of important observations. First of all, in the technical literature, there is a scarcity of reports about reconciliation among all three software development models analyzed.
models. By the end of the review, only a single paper (Theunissen et al., 2008) had been found covering all of them. Theunissen et al. (2008) discuss the challenges for reconciliation between agile and FOSS development models within a corporate culture. The type of reconciliation considered in the paper concerns an agile team working within an organization and collaborating with an outside FOSS project. In fact, the paper does not exactly focus on the three software development models, since the plan-driven view is not embodied in the process, but in the environment and organizational culture. In addition, this reconciliation does not happen with the three models being used within the same organization.

This raises the question: why are there so few studies correlating the three software development models? After some reflection, the first possibility to be considered is that not all authors and researchers see a development process in FOSS projects, considering that these projects are conducted in an ad-hoc manner. However, as discussed in Section 2, we do not support this idea and have decided to look for other alternatives.

The most reasonable alternative answer to this question is that reconciling only two software development models is already a major challenge. This research issue is quite recent, given that preliminary research on reconciling models was published in 2002, describing proposals to combine only two models, before engaging a third complicating factor in this equation.

This hypothesis is also underpinned by a second observation: there is more work to combine the plan-driven and agile models. Since, for some time, the plan-driven model was the established standard in many organizations, it is the starting point for many reconciliation initiatives. Therefore, that effort to combine plan-driven and agile models should predominate is justified by the fact that both are in widespread use in organizations and there is a common sense in the field that agile and plan-driven approaches are poles apart, which makes them a focus and motivates research. Therefore, this could be a progressive way to reach reconciliation of others.

Another observation is that reconciliation between the agile and FOSS development models is still incipient. In general, it is based on integrating one particular practice, like test-driven development into a FOSS project. Although this combination is not yet widely explored in literature, some authors (Simmons and Dillon, 2003; Warsta and Abrahamsson, 2003) emphasize the proximity and similarities of these two models and consider that the FOSS model may lie between the plan-driven and agile models (Abrahamsson et al., 2002). For example, both agile and FOSS approaches emphasize iterative development and frequent releases, with minimum documentation and a focus on testing (Barnett, 2004). However, they differ in one significant respect: several agile methods emphasize co-location and face-to-face development, while FOSS typically involves geographically distributed developers. In addition, FOSS needs tools to make its distributed approach work, while agile methods do not. Therefore, FOSS is similar to plan-driven in all the respects it is dissimilar to agile. Other similarities and differences between the two models are described by Tsirakidis et al. (2009).

Finally, the results of this quasi-systematic review also indicate that organizations can reconcile the three development models at three different levels: organizational, group, or process. At the organizational level, the organization can have independent development teams, segregated in different organizational units, each of which uses one of the three development models. This allows the organization to work with complete as well as hybrid methods, and does not pose the organizational culture challenge. At the group level, each development team can switch between different models depending on the project context. This assumption that any organizational culture issues have been overcome and both complete and hybrid models can be used. At the process level, a new hybrid process can be created by combining various parts of each method. This only allows hybrid methods, and assumes that cultural challenges can be overcome to some extent. These levels will be used to categorize the proposals (SQ2), strategies (SQ3) and experiences (SQ4).

7.2. SQ1: what are the opportunities and challenges to reconciliation among plan-driven, agile, and FOSS software development models?

To answer this secondary question, we analyzed opportunities and challenges separately. Accordingly, the 14 papers identified (Table 6) for SQ1 are distributed in Tables 7 and 8. Note that these opportunities and challenges refer to reconciliation among models, since this study is not concerned to investigate the potential benefits or problems of individual models.

The main opportunities for reconciling plan-driven, agile, and FOSS development models are summarized in Table 7 and ordered according to the number of citations. These results indicate an interest in collaboration. Reconciliation can stimulate people’s involvement in the development process, allowing them to meet and collaborate to facilitate group efforts in the future. Also, as regards reconciliation of FOSS with plan-driven or agile development, there is also the possibility of collaborating with open source communities with large pools of developers available.

In addition, availability of code is seen as an opportunity for innovation in software development. Other opportunities mentioned, such as quality improvement and cost reduction, are common arguments for research into software engineering processes and software development in general.

The main challenges to reconciliation among the three software development models are summarized in Table 8, which shows a diversity of problems that organizations and researchers must address to make reconciliation feasible. Not surprisingly, the most cited challenge is the barrier to changing organizational culture. When teams are used to working with one development model or when the organization has a culture that favors practices from one specific model, reconciliation can be a major challenge.

The need to review the number of artifacts usually created during a plan-driven project is also pointed to as an important challenge to reconciliation, since agile and FOSS development emphasize simplicity and parsimony in documentation. Furthermore, in plan-driven development, knowledge is considered a scarce resource that must be protected, and a source of power. Thus, it is hardly shared, but restricted to certain participants. FOSS projects, on the other hand, are based on the free exchange of knowledge through information and communication technology.

7.3. SQ2: what proposals have been made for reconciliation among plan-driven, agile, and FOSS software development models?

The main proposals for reconciling software development models have been categorized and are presented in Table 9. Although each proposal is briefly summarized, it is not the purpose of this section to discuss their validity or importance, since this is an exploratory study to achieve a better comprehension of the research field.

The most common type of proposal is one that suggests adapting reference models – such as the staged model for software evolution (Capiluppi et al., 2007), PROFES (Product Focused Improvement of Embedded Software Processes) (Jedlitschka et al., 2005) or Six Sigma (Hashmi and Jongmoon Baik, 2008) – to introduce agile or FOSS development practices into specific contexts like COTS reuse (Navarrete et al., 2007), distributed development (Kornstädt and Sauer, 2007), or mobile applications (Rahimian and Ramsin, 2008).

Table 7
Key opportunities for reconciliation among software development models.

<table>
<thead>
<tr>
<th>#</th>
<th>Opportunities</th>
<th>Description</th>
<th>Papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Collaboration</td>
<td>Facilitates collaboration among participants in the process or offers the possibility of collaborating with open source communities with large pools of developers available</td>
<td>[5,12,30]</td>
</tr>
<tr>
<td>2.</td>
<td>Availability of code</td>
<td>Availability and transparent access to source code favors innovation</td>
<td>[12,14]</td>
</tr>
<tr>
<td>3.</td>
<td>Cost</td>
<td>Reducing project costs</td>
<td>[12,25]</td>
</tr>
</tbody>
</table>

In proposals of this kind, adaptation focuses on the whole model and aims to produce a new hybrid model designed for a specific context.

Application of specific practices, such as configuration management (Asklund and Bendix, 2002), continuous code integration (Deshpande and Riehle, 2008), knowledge management (Awazu and Desouza, 2004), test-driven development (Turnu et al., 2006), and peer review (Johnson, 2006), from one model to another is also a widespread kind of approach, probably because it reduces the reconciliation problem to a particular practice. In this case, it is easier to plan, implement, monitor and control the results of such reconciliation. This kind of proposal does not aim to define a new hybrid model. It investigates the effect of using these individual practices in a project guided by a different model.

It can be seen from Table 9 that this application of practices usually involves two development models. Asklund and Bendix (2002) compare configuration management (CM) in plan-driven and FOSS contexts in order to understand the best practices applied in each and to discuss how these can be transferred between them to mutual benefit. They do not discuss CM in an agile context. This makes sense because agile methods do not address any specific practice to CM and do not emphasize the use of tools, which are very important to CM work. On the other hand, agile is concerned with adapting to changes and product integrity and, therefore, some CM practices are also needed. Accordingly, agile would also benefit from the analysis and transfer of some plan-driven and FOSS CM practices, although these practices would need some adaptation, as discussed in Moreira (2009).

Continuous code integration, a common practice in the agile world, was also investigated in the context of FOSS projects (Deshpande and Riehle, 2008). Their results are inconclusive, as it is not possible to say whether FOSS projects do not adopt continuous code integration on a significant scale or FOSS projects have always practiced continuous integration. In this case, the advent of agile methods did not bring any changes, simply because they had arrived in open source software development long before. There is thus a need for research on this topic to be broadened and extended to the context of plan-driven projects.

FOSS communities were also examined for insights into how to expand current knowledge management practices in organizations (Awazu and Desouza, 2004). In traditional knowledge management, knowledge is considered a valuable, rare, inimitable, and non-substitutable organizational resource that contributes to sustained competitive advantage. In this mindset, knowledge is a scarce resource, so it must be protected. On this view, organizations share knowledge only on a need-to-know basis. The rigid structure of knowledge management discourages innovation and offers poor incentives to reuse knowledge. However, these problems traditionally encountered by knowledge management efforts in organizations are, for the most part, absent in FOSS communities. FOSS communities have large populations of developers, who

Table 8
Key challenges for reconciliation among software development models.

<table>
<thead>
<tr>
<th>#</th>
<th>Challenges</th>
<th>Description</th>
<th>Papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Organizational culture</td>
<td>Cultural barriers – customers, managers, and developers may not accept development model change</td>
<td>[1,20,24,29,42]</td>
</tr>
<tr>
<td>2.</td>
<td>Number of artifacts</td>
<td>The reputation for copious documentation (plan-driven) contrasts with the simplicity and parsimony of creating only the really necessary artifacts (FOSS and agile)</td>
<td>[29,37,39,42]</td>
</tr>
<tr>
<td>3.</td>
<td>Knowledge management</td>
<td>Explicit knowledge (plan-driven) versus tacit knowledge (FOSS and agile)</td>
<td>[14,39]</td>
</tr>
<tr>
<td>4.</td>
<td>Organizational structure</td>
<td>Hierarchical control (plan-driven) versus flexibility on a voluntary basis (FOSS) or self managed teams (agile)</td>
<td>[24,39]</td>
</tr>
<tr>
<td>5.</td>
<td>Recognition and reward system</td>
<td>The organization recognition and reward system is usually based on individual assessment (plan-driven and some FOSS projects) and must be reviewed to stimulate collaboration and teamwork</td>
<td>[24,39]</td>
</tr>
<tr>
<td>6.</td>
<td>Communication</td>
<td>Face-to-face communication (agile) versus communication over the Internet and distributed development (FOSS and plan-driven)</td>
<td>[1,20]</td>
</tr>
<tr>
<td>7.</td>
<td>Decision making</td>
<td>Centralized (plan-driven) versus decentralized decision making (FOSS and agile)</td>
<td>[1,24]</td>
</tr>
<tr>
<td>8.</td>
<td>Quality assurance</td>
<td>Peer review (FOSS and agile) versus formal techniques that make non-conformances visible to senior management (plan-driven)</td>
<td>[1,25]</td>
</tr>
<tr>
<td>9.</td>
<td>Requirements</td>
<td>Complete and documented requirements (plan-driven) versus functional and informal (FOSS and agile) requirements</td>
<td>[31,37]</td>
</tr>
<tr>
<td>10.</td>
<td>Planning work</td>
<td>Documented and formal planning (plan-driven) versus informal planning (FOSS and agile)</td>
<td>[1]</td>
</tr>
<tr>
<td>11.</td>
<td>Monitoring work</td>
<td>Monitoring activities carried out by developers (plan-driven) versus voluntary work (FOSS) or teams controlling their own tasks (agile)</td>
<td>[1]</td>
</tr>
<tr>
<td>12.</td>
<td>Customer relationship</td>
<td>Need to take actions to prepare the client for the new development process</td>
<td>[39]</td>
</tr>
<tr>
<td>13.</td>
<td>Technology</td>
<td>Appropriateness of existing technologies, environments and tools to deployment of the new development model</td>
<td>[39]</td>
</tr>
<tr>
<td>14.</td>
<td>Keeping certified processes</td>
<td>Organizations are concerned to improve their software development processes and, at the same time, to maintain certification of their processes (plan-driven)</td>
<td>[31]</td>
</tr>
</tbody>
</table>
share common interests and backgrounds, and are available to give rapid feedback and immediate solutions to specific problems. They also have experts motivated by the idea of forming part of a core group and gaining ultimate peer recognition. Although not mentioned in the paper, in agile development a great deal of product knowledge is tacit and shared among project members through face-to-face communication and personal interactions. It would thus also be interesting for future studies to investigate how these agile practices can contribute to improving knowledge management in organizations or in FOSS communities. In addition, the use of explicit knowledge management practices, coming from plan-driven and FOSS development, should be carefully analyzed to ascertain whether it shows any potential to be beneficial in an agile context.

As regards test driven development, the goal of the work is to study the effects of adopting agile practices in FOSS development (Turnu et al., 2006). In particular, Turnu et al. started evaluating test driven development because they considered it easier to apply in a distributed environment than most other agile practices. Using a simulation modeling approach, they concluded that this practice yields better results in terms of code quality. However, real case studies are necessary to confirm these results and to extrapolate the use of this practice to a plan-driven context.

Another type of proposal presented in literature combines practices from different models, such as XP or Scrum with CMMI or ISO 12207 (Marcel et al., 2007; Vriens, 2003). This generally involves mapping equivalence between practices from these models. It results in an analysis of how agile methods adhere to plan-driven maturity models and how agile methods can be modified to incorporate the practices that are missing. This type of reconciliation proposal usually does not consider FOSS development, because it has no formal model or complete definition that can be used to compare and fit agile methods or plan-driven reference models. Although this kind of proposal also results in definition of a hybrid model, it differs from the adaptation of reference models, because the model here is generic to software development and not specific to a context like COTS or mobile.

However, the complex nature of software development and the wide variety of existing models (XP, Scrum, Crystal, FDD, Lean, CMMI, ISO 12207, ISO 9001 – to mention just some) make this comparison an arduous and imprecise task (Boehm and Turner, 2003b). Furthermore, this kind of integration among development models limits the potential for future synergy among them, because it builds an incomplete hybrid model, and it cannot be guaranteed that the resulting process actually embodies the characteristics or philosophies of the original models.

The proposal of Anderson (2005) also refers to the process level, but has a different focus. This proposal is based on the idea that maturity models are not prescriptive and their practices are only indicative. Therefore, it is possible to use alternative practices that facilitate compatibility with agile methods. The key point is to provide appropriate evidence of a practice’s effective implementation in an organization. This interpretation is not always applied in certified organizations. It can be done using Deming’s notion that quality refers to conformance to process. This conformance can be checked through an understanding of natural variation in processes and by statistical analysis to verify that processes are under control (see more details in Florac and Carleton, 1999), considering metrics that make sense in both the plan-driven and agile worlds. The use of FOSS processes is not discussed, but these principles can be applied to any development model.

Boehm and Turner’s (2003a, 2004) proposal suggests risk analysis of project characteristics as a way to select the appropriate method and achieve balance between agility and discipline. This analysis considers three categories of risk: environment risks, the risks of adopting an agile model, and the risks of adopting a plan-driven model. On the citations map (Fig. 4) this proposal proved to be the most referenced by other researchers.


Table 10
Strategies for reconciling software development models in organizations or communities.

<table>
<thead>
<tr>
<th>#</th>
<th>Organizations or communities</th>
<th>Levels</th>
<th>Strategies</th>
<th>Positive results</th>
<th>Risks</th>
<th>Papers</th>
</tr>
</thead>
</table>
| 1. | ABB automation and ericsson microwave | Process | Adaptation of the stage-gate model to incorporate agile practices from XP. | + Early feedback  
+ Important features not delayed  
+ Micro-planning avoids rigid plans  
+ Internal communication improved  
+ Product quality  
+ Concrete measures of development project progress  
+ Developer motivation increased  
+ Exploration of customer quality requirements  
+ Development of a learning organization  
+ Available resources invested in product quality-related processes in a goal-oriented manner in line with organizational needs  
+ Customer integration  
+ Decision based on project characteristics | – Little long-term planning  
– Agile team isolation within the organization  
– Managers feel threatened  
– Training needs | [26] |
| 2. | BMW Car IT                    | Process | Adaptation of a traditional process improvement model (PROFES) for agile development, combining XP, Scrum and FDD | + Exploration of customer quality requirements  
+ Development of a learning organization  
+ Available resources invested in product quality-related processes in a goal-oriented manner in line with organizational needs  
+ Customer integration  
+ Decision based on project characteristics | – Customer unable to articulate quality requirements explicitly | [17] |
| 3. | DTE Energy                    | Process | Use in-house agile initiative for process improvement designed to combine CMMI Level 3 with agile methods | Paper reports IT organization's journey towards combining CMMI and agile, but does not present later results. Summarizes some recommendations for other organizations interested in taking this approach | | [29] |
| 4. | Memphis University             | Group   | Experience report on proposed risk analysis that resulted in choice of XP for development project in question. | + Develops a truly agile full life-cycle process  
+ Meets requirements for all 5 CMMI model levels  
+ Adheres to Agile Manifesto  
+ Process heaviness was reduced by around 83%  
+ Defines metrics useful to agile development | – Lack of team training and experience | [28] |
<p>| 5. | Microsoft                     | Process | Use of statistical control for integrating CMMI level 5 with agile development. | | – Hard work to process design | [38] |</p>
<table>
<thead>
<tr>
<th>Process Name</th>
<th>Process Description</th>
<th>Benefits</th>
<th>Challenges</th>
</tr>
</thead>
</table>
| Motorola Wireless | Combination of CMMI level 5 with agile methods | * Improved quality - defects reduced 75%  
* Improved cycle time  
* Reduced risk  
* Productivity improved threefold  
* Staff effort reduced 59.6% | - Agile scaling up  
- Potential for ambiguous requirement statements if customer and team fail to collaborate  
- Customer cannot be on-site full time in large distributed projects  
- Team members and customers may not accept process paradigm shift |
| Philips | Combination of CMM level 2 and ISO 9001 with XP and Scrum | * ISO 9001 Certification | - Lack of senior management involvement  
- Lack of agile training and mentoring  
- Difficult to implement and to maintain focus on using the practice |
| PyPy community | FOSS project using an agile practice | * Technical progress | - Difficult to implement and to maintain focus on using the practice |
| Systematic | Combination of Scrum and CMMI models | Make recommendations to activities an agile project could consider adopting. Activities inspired by mandatory goals and expected practices from subset of CMMI process areas | - Difficult to educate user representatives on agile methods |
| U.S. Army | Combination of CMII level 5 with XP, selecting best practices for institutionalization | * Not blind adoption of all agile approach techniques  
* Best method depends on nature of project Findings discussed by key issue: management and organizational, people, processes and technology (see details in paper) | |
| Volvo Technology | Combination of ISO 15504 and XP models | | |
The organizational structure proposal (Vinekar et al., 2006) is the only one that refers to organizational level. It attempts to accommodate the organizational culture of plan-driven and agile software development by separating development models into different organizational units subordinated to a common management. It separates the development teams of different development models, and each project is developed in the unit with characteristics that fit the project most closely. The idea of organizational separation can also be applied to create a unit dedicated to FOSS development.

Finally, the project improvement framework proposal (Reifer, 2003) argues that CMM can be used to introduce XP practices in organizations. The infrastructure and culture of CMM processes can help institutionalize XP. In addition, CMM management practices may also contribute to overcoming XP scalability limitations. Even though Reifer claims that these points could help reconciliation, he acknowledges that it is not what is happening in practice and reports some resistance to change (which is a major challenge discussed for several authors, as presented in SQ1).

7.4. SQ3: what strategies are adopted by organizations or communities for reconciliation among plan-driven, agile, and FOSS software development models?

7.4.1. SQ4: what results have been achieved by organizations or communities that have attempted reconciliation among plan-driven, agile, and FOSS software development models?

The answers to SQ3 and SQ4 were consolidated and are presented in Table 10, which shows a significant number of large companies interested in this topic. The Levels column indicates the levels at which reconciliation (organizational, group, or process) was pursued. As can be seen, almost all the strategies adopted focus on the process level. The predominant strategy adopted is a combination of practices from different models, like CMMI or ISO with XP and Scrum.

Interestingly, in the scope of this review, experiences with reconciliation at the organizational level were not identified. This can indicate that organizations are not investing on separating teams by development model. There is only one experience report, of risk analysis using Boehm and Turner’s (2003a, 2004) proposal, that pursued reconciliation at the group level. After an analysis of project characteristics, a pure agile approach was used. However, this decision was not based on segregated organizational units, as the same team was prepared to work with both plan-driven and agile methods. Agile choice was determined considering project needs, which appears to be a promising path to reconciliation in the future.

Notice also that only one experience reports reconciliation of FOSS. PyPy is a FOSS project that adopted an agile practice – sprint-oriented development – to address the problem of distributed development.

The main achievements from the organizations’ strategies were separated (above) into positive results or risks. Although the reports tend to emphasize the positive results, they seem to suggest that model reconciliation is a viable approach and a good strategy for achieving better quality and productivity results. Some of the risks identified echo the main challenges cited in SQ1. Finally, some of the studies (as in the case of DTE Energy) do not analyze the results objectively, but provide recommendations for how to conduct the reconciliation in practice.

8. Conclusions

This paper presented a quasi-systematic review to characterize reconciliation among software development models. Systematic reviews are useful to identify and consolidate work to date and guide future research. The conclusions reached are an important step towards expanding the body of knowledge about reconciliation among software development models. Based on these results, some directions for future research are outlined in Section 8.2.

The main conclusion of this study is that there are three different levels of reconciliation: organizational, group, and process. Each level requires different approaches for merging methods and/or facing cultural issues.

The results also show that, to date, few studies have combined plan-driven, agile, and FOSS development models. Moreover, the larger number of works combines plan-driven and agile models. Reconciliation between agile and FOSS models is still incipient and generally based on integrating one particular practice in the context of a FOSS development project.

The main opportunities for reconciliation derive from collaboration and code availability. Collaboration can stimulate people’s involvement in the development process and offers the possibility of interacting with open source communities with large pools of available developers. The code availability favors innovation in software development.

Proposals are most commonly of three types: (i) adapting traditional reference models by introducing agile or FOSS practices in specific contexts; (ii) proposals for the transfer of specific practices from their original model to another are also widespread, probably because they simplify the reconciliation problem to a particular practice; and (iii) the combination of practices from different models, comparing and mapping equivalences between practices from these models.

Finally, we detected keen interest in reconciliation on the part of large organizations. However, they are attempting to combine typically plan-driven and agile models, without the help of detailed guidelines. Their reports indicate that reconciliation among models is a viable approach and a rewarding strategy for achieving better quality and productivity results.

Despite existing proposals, reconciliation among software development models to achieve a balanced mix of plan-driven, agile, and FOSS is still an open issue, because it is an emerging area and research on most of the proposals is at an early stage. There is a diversity of challenges that organizations and researchers must address to make reconciliation feasible. The most salient is to overcome barriers to culture change. The number of artifacts usually created during a plan-driven project is also a point of divergence, as agile and FOSS development emphasize simplicity and parsimony in documentation. We argue that knowledge of these challenges can help the development of strategies to overcome them in future.

We believe that this characterization is important for practitioners who want to stay current with the state of research. This review will also assist the scientific community working with software development processes to build a common understanding of the challenges that must be addressed, as well as to identify topic areas that have been researched or where research is still lacking. Finally, the results of this kind of investigation are relevant to software industry, which is calling for solutions in this area.

8.1. Limitations

This section discusses the main limitations of this research, including the threats to validity already mentioned in Section 6.1.

The first limitation refers to the automated search used. Although this type of broad, automated search tends to find more prominent studies (Kitchenham et al., 2009a), it is also restricted by the standing of the publication, i.e., even relevant papers cannot be captured if they appeared in publications that are not indexed. This limitation can be overcome by the use of manual searches in major conferences, at least to complement the results of the automated
search. The unavailability of a few papers also prevents them from contributing to the topic, even if they have potential to do so.

Data sources such as Google Scholar, and academic theses and dissertations on the topic, are – perhaps conspicuously – not used by this study. As explained in Section 4.4, we selected data sources that allowed us to make automated searches of their content, and this was not the case with Google Scholar or the body of knowledge contained in theses and dissertations. The protocol could not be executed in Google Scholar, which even in advanced search mode does not admit the whole search string. Although to date there is no database that indexes theses and dissertations and can be used for automated searches, such theses and dissertations often do generate papers that can be captured in the databases that were used.

Finally, as mentioned in Section 7, as an emerging field, there is a scarcity of works addressing reconciliation among plan-driven, agile and FOSS development models. Even so, this paper offers many important observations that represent a significant starting point for future research on this topic, as presented in the following section.

8.2. Directions for future research

The first opportunity for future research lies in re-execution of the protocol, to capture references to more recent work that extends the search space chronologically. This could also include other search engines, such as ACM (Association for Computing Machinery), in an attempt to retrieve documents only indexed by these machines, which would extend the search space geographically. Finally, the search can also be expanded with manual searches to include: books; conferences; theses and dissertations; technical reports; and other search engines, such as Google Scholar and AISel (Association for Information Systems Electronic Library). Although the systematic approach adopted ensures the reliability and completeness of this study, it can be amplified by these extensions. In Section 7 (SQ2), we discussed how studies that showed how to combine two development models could be extended to the third. This discussion included ideas like: (i) the reconciliation of FOSS and plan-driven configuration management practices to agile model; (ii) the need for further studies on the reconciliation of the practice of continuous code integration between agile and FOSS models and extension of this search to the context of the plan-driven model; (iii) investigate how the knowledge management practices in agile model can contribute to improve knowledge management in organizations or in FOSS communities; (iv) analyze if the use of explicit knowledge management practices, coming from plan-driven and FOSS development, can be beneficial in an agile context; (v) extrapolate the use of test driven development to a plan-driven context. These ideas are all opportunities for future research. An area that still deserves to be explored is the search for studies to reconcile the three models of software development, since only one study was identified in this quasi-systematic review. First of all, it can be present in areas that were not the focus of this work. Some of these understudied areas are indicated below to guide future research. In addition, as stated before, the reconciliation research area is still at an early stage, but it is expected that in future there are more studies and results on the topic to be investigated. Finally, there also remains the possibility that organizations or other researchers have already achieved positive results on the reconciliation among agile, FOSS, and plan-driven models, but have not written about it yet.

Another interesting topic that deserves attention in future research is the emerging body of literature on agile and distributed development (Angioni et al., 2005; Ramesh et al., 2006). The coming together of these two worlds was not explored in this study, but we consider that they can also contribute to facilitate reconciliation among plan-driven, agile and FOSS development models. Likewise, the idea of processes diversity (where different processes may be running concurrently on the same project—in multi-team projects or changing over time during the phases of the development and maintenance cycle) (Deck, 2001; Lindwall and Rus, 2000; Siebel et al., 2003) also should be investigated in future research, since the need to manage this diversity can be an important motivation for the reconciliation among software development models. Finally, approaches that deal with reconciliation considering organization and project contexts and needs, appears to be a promising path for reconciliation in the future (Jaufman and Munch, 2005; Xu and Ramesh, 2008). This is the focus of the next section.

8.3. Future work

Our research agenda on this topic involves investigating software development process tailoring (Pedreira et al., 2007) as a way to achieve a balance between characteristics of software development models. Process tailoring is the act of particularizing a general process description to derive a new process applicable to a specific situation (Ginsberg and Quinn, 1995).

We claim that it is necessary to tailor these processes to suit specific project and organization contexts. This tailoring should also consider the key distinctive features of plan-driven, agile and FOSS models: collaboration and discipline (Magdaleno, 2010a). Collaboration can be defined as a group of two or more people working to achieve a common goal (Vrede and Briggs, 2005). Collaboration is an important factor for software organizations to achieve their productivity, quality and knowledge-sharing goals. In particular, software development is a complex process that involves collaboration among several people over time to achieve a common goal (Cugola and Ghezzi, 1998). Therefore, software development is a typical example of collaborative work (DeMarco and Lister, 1999). Discipline, meanwhile, relates to the planning level adopted in software process definition and the rigidity of control employed in process execution (Boehm and Turner, 2003b).

Both are complementary and essential in any project, but in differing proportions, depending on project characteristics. For a balanced mix between collaboration and discipline, it is necessary to understand how these aspects vary and distinguish the software development models (Magdaleno et al., 2010). In order to facilitate process tailoring, it is possible to support the project manager by automating some of the steps, possibly reducing the effort required to conduct this activity and improving the quality and appropriateness of the resulting process (Park et al., 2006). Thus, we intend to develop an infrastructure using an optimization-based approach (Magdaleno, 2010b). Finally, it is also important to investigate whether the results of this quasi-systematic review are consistent with what is observed in practice, i.e., in organizational routine. To accomplish this, we intend to plan and conduct an experimental study. This experimental study will involve a survey of industry and academic experts to validate the results and discuss the conclusions. A similar strategy was applied by (Bekkers et al., 2008) to determine the factors that most influence the selection of software development model.

Acknowledgments

This research was partially funded by CNPq under process No. 142006/2008-4 and No. 310776/2009-0. The authors would like to thank: Prof. Guilherme Travassos for his careful revision of the protocol; Vanessa Nunes (PhD student), Taisa Santos, and Alice Leite (MSC students), who helped validate this quasi-systematic review; and anonymous reviewers whose comments significantly contributed to the quality of this work.
Appendix A. Extraction form

Table 11 presents the information extraction template.

<table>
<thead>
<tr>
<th>#</th>
<th>Question</th>
<th>Title and reference</th>
<th>Library</th>
<th>Inclusion criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>MQa</td>
<td>Corporate-, agile- and open source software development: a witch’s brew or an elixir of life? (Theunissen et al., 2008)</td>
<td>Compendex</td>
<td>IC1, IC2, IC3</td>
</tr>
<tr>
<td>2.</td>
<td>MQb</td>
<td>Agile principles and open source software development: a theoretical and empirical discussion (Koch, 2004)</td>
<td>WebOfScience</td>
<td>IC1</td>
</tr>
<tr>
<td>3.</td>
<td>MQb</td>
<td>Continuous integration in open source software development (Deshpande and Riehle, 2008)</td>
<td>Scopus</td>
<td>IC1, IC3</td>
</tr>
<tr>
<td>4.</td>
<td>MQb</td>
<td>Modeling and simulation of open source development using an agile practice (Tunu et al., 2006)</td>
<td>Scopus</td>
<td>IC1, IC2, IC3</td>
</tr>
<tr>
<td>5.</td>
<td>MQb</td>
<td>Sprinting towards open source development (Goth, 2007)</td>
<td>Scopus</td>
<td>IC1, IC2, IC4</td>
</tr>
<tr>
<td>6.</td>
<td>MQc</td>
<td>A development process for building OSS-based applications (Huang et al., 2006)</td>
<td>WebOfScience</td>
<td>IC1, IC3</td>
</tr>
<tr>
<td>7.</td>
<td>MQc</td>
<td>A framework for creating, hybrid-open source software communities (Sharma et al., 2002)</td>
<td>WebOfScience</td>
<td>IC1, IC3</td>
</tr>
<tr>
<td>8.</td>
<td>MQc</td>
<td>A study of configuration management in open source software projects (Asklund and Bendix, 2002)</td>
<td>Compendex</td>
<td>IC1, IC3</td>
</tr>
<tr>
<td>9.</td>
<td>MQc</td>
<td>Achieving quality in open-source software (Aberdour, 2007)</td>
<td>Compendex</td>
<td>IC1, IC3</td>
</tr>
<tr>
<td>10.</td>
<td>MQc</td>
<td>Adapting the “staged model for software evolution” to free/libre/open source software (Capioluppi et al., 2007)</td>
<td>Compendex</td>
<td>IC1, IC3</td>
</tr>
<tr>
<td>11.</td>
<td>MQc</td>
<td>Collaboration, peer review and open source software (Johnson, 2006)</td>
<td>WebOfScience</td>
<td>IC1, IC3</td>
</tr>
<tr>
<td>12.</td>
<td>MQc</td>
<td>Commercial adoption of open source software: an empirical study (Glynn et al., 2005)</td>
<td>Compendex</td>
<td>IC1, IC2</td>
</tr>
<tr>
<td>13.</td>
<td>MQc</td>
<td>Open collaboration within corporations using software forges (Riehle et al., 2009)</td>
<td>IEEE</td>
<td>IC1</td>
</tr>
<tr>
<td>15.</td>
<td>MQc</td>
<td>Open source software—an evaluation (Fuggetta, 2003)</td>
<td>WebOfScience</td>
<td>IC1</td>
</tr>
<tr>
<td>16.</td>
<td>MQd</td>
<td>A soft-structured agile framework for larger scale systems development (Soundararajan and Arthur, 2009)</td>
<td>IEEE</td>
<td>IC1, IC3</td>
</tr>
<tr>
<td>17.</td>
<td>MQd</td>
<td>Adapting PROFES for use in an agile process: an industry experience report (Jedlitschka et al., 2005)</td>
<td>Compendex</td>
<td>IC1, IC3, IC4</td>
</tr>
<tr>
<td>18.</td>
<td>MQd</td>
<td>Addressing diverse needs through a balance of agile and plan-driven software development methodologies in the core software engineering course (Layman et al., 2008)</td>
<td>WebOfScience</td>
<td>IC1, IC3</td>
</tr>
<tr>
<td>19.</td>
<td>MQd</td>
<td>Agile development: Good process or bad attitude? (Turner, 2002)</td>
<td>WebOfScience</td>
<td>IC1, IC3</td>
</tr>
<tr>
<td>20.</td>
<td>MQd</td>
<td>Applying agile principles for distributed software development (Phamikar et al., 2009)</td>
<td>IEEE</td>
<td>IC1, IC2</td>
</tr>
<tr>
<td>21.</td>
<td>MQd</td>
<td>Army simulation program balances agile and traditional methods with success (Surd and Parsons, 2006)</td>
<td>Compendex</td>
<td>IC1, IC4</td>
</tr>
<tr>
<td>22.</td>
<td>MQd</td>
<td>Balancing agility and discipline: evaluating and integrating agile and plan-driven methods (Boehm and Turner, 2004)</td>
<td>Compendex</td>
<td>IC1, IC3</td>
</tr>
</tbody>
</table>

Appendix B. Studies included in the review

Table 12 lists the 42 papers selected for this quasi-systematic review. The following information was gathered for each publication: title, database from which this publication was selected, and the inclusion criteria applied.

Table 12
Papers selected in the review.

Table 12 (Continued)

<table>
<thead>
<tr>
<th>#</th>
<th>Question</th>
<th>Title and reference</th>
<th>Library</th>
<th>Inclusion criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>23.</td>
<td>MQd</td>
<td>Balancing the human and the engineering factors in software development (Minkandla and Dwolatzky, 2004)</td>
<td>Compendex</td>
<td>IC1, IC3, IC4</td>
</tr>
<tr>
<td>24.</td>
<td>MQd</td>
<td>Can agile and traditional systems development approaches coexist? An ambidextrous view (Vinekar et al., 2006)</td>
<td>WebOfScience</td>
<td>IC1, IC2, IC3</td>
</tr>
<tr>
<td>25.</td>
<td>MQd</td>
<td>Certifying for CMMI Level 2 and ISO9001 with XP@Scrum (Vriens, 2003)</td>
<td>IEEE</td>
<td>IC1, IC2, IC4</td>
</tr>
<tr>
<td>26.</td>
<td>MQd</td>
<td>Combining agile methods with stage-gate project management (Karlstrom and Runeson, 2005)</td>
<td>Compendex</td>
<td>IC1, IC4</td>
</tr>
<tr>
<td>27.</td>
<td>MQd</td>
<td>Designing an agile methodology for mobile software development: a hybrid method engineering approach (Rahman and Ramun, 2006)</td>
<td>IEEE</td>
<td>IC1, IC3</td>
</tr>
<tr>
<td>28.</td>
<td>MQd</td>
<td>Experiences using agile software development for a marketing simulation (Mills et al., 2006)</td>
<td>IEEE</td>
<td>IC1, IC4</td>
</tr>
<tr>
<td>29.</td>
<td>MQd</td>
<td>Formalizing agility, part 2: how an agile organization embraced the CMMI (Baker, 2006)</td>
<td>Compendex</td>
<td>IC1, IC2, IC4</td>
</tr>
<tr>
<td>30.</td>
<td>MQd</td>
<td>Future implementation and integration of Agile methods in software development and testing (Kee, 2006)</td>
<td>Compendex</td>
<td>IC1, IC2, IC4</td>
</tr>
<tr>
<td>31.</td>
<td>MQd</td>
<td>Management challenges to implementing agile processes in traditional development organizations (Boehm and Turner, 2005)</td>
<td>Compendex</td>
<td>IC1, IC2</td>
</tr>
<tr>
<td>32.</td>
<td>MQd</td>
<td>Mapping CMMI project management process areas to SCRUM practices (Marcal et al., 2007)</td>
<td>Compendex</td>
<td>IC1, IC3</td>
</tr>
<tr>
<td>33.</td>
<td>MQd</td>
<td>Mastering dual-shore development—the tools and materials approach adapted to agile offshoring (Kornstädt and Sauer, 2007)</td>
<td>Compendex</td>
<td>IC1, IC3</td>
</tr>
<tr>
<td>34.</td>
<td>MQd</td>
<td>Mature agile with a twist of CMMI (Jakobsen and Johnson, 2008)</td>
<td>Compendex</td>
<td>IC1, IC3, IC4</td>
</tr>
<tr>
<td>35.</td>
<td>MQd</td>
<td>Quantitative process improvement in XP using six sigma tools (Hashimi and Jongmoon Baik, 2008)</td>
<td>Compendex</td>
<td>IC1, IC3</td>
</tr>
<tr>
<td>36.</td>
<td>MQd</td>
<td>Reconciling agility and discipline in COTS selection processes (Navarrete et al., 2007)</td>
<td>Compendex</td>
<td>IC1, IC3</td>
</tr>
<tr>
<td>37.</td>
<td>MQd</td>
<td>Requirements engineering and agile software development (Paetsch et al., 2003)</td>
<td>IEEE</td>
<td>IC1, IC2, IC3</td>
</tr>
<tr>
<td>38.</td>
<td>MQd</td>
<td>Stretching agile to fit CMMI level 3—the story of creating MSF for CMMI® process improvement at Microsoft corporation (Anderson, 2005)</td>
<td>Compendex</td>
<td>IC1, IC3</td>
</tr>
<tr>
<td>39.</td>
<td>MQd</td>
<td>The role of extreme programming in a plan-driven organization (Dahlberg et al., 2006)</td>
<td>Scopus</td>
<td>IC1, IC2, IC4</td>
</tr>
<tr>
<td>40.</td>
<td>MQd</td>
<td>Using risk to balance agile and plan-driven methods (Boehm and Turner, 2003a)</td>
<td>Compendex</td>
<td>IC1, IC3</td>
</tr>
<tr>
<td>41.</td>
<td>MQd</td>
<td>When software engineers met research scientists: A case study (Segal, 2005)</td>
<td>Compendex</td>
<td>IC1</td>
</tr>
<tr>
<td>42.</td>
<td>MQd</td>
<td>XP and the CMMI (Reifer, 2003)</td>
<td>Compendex</td>
<td>IC1, IC2, IC3</td>
</tr>
</tbody>
</table>

References


Cockburn, A. 2001. Agile Software Development. Addison-Wesley, Boston, MA, USA.


Andrêa Magalhães Magalhães is a D.Sc. student at the Computer Science Department of COPPE/UFRJ, Brazil. She obtained her M.Sc. degree in computer science at UFRJ in 2007. She is the author of publications in Brazilian and international conferences and journals. She coordinates consulting projects at companies in the field of business process modeling and requirements engineering. Her areas of interest are: software engineering, collaboration, and business process management.

Renata Mendes de Araujo obtained her D.Sc. in computer science from COPPE/UFRJ (2000). She is an associate professor at UNIRIO (Rio de Janeiro State Federal University), Brazil, and a CNPq researcher. Her experience and research work focus on information systems, software engineering, software development process improvement, business process management, computer-supported cooperative work and e-democracy. She coordinates consulting projects in the field of software development and business process modeling.

Cláudia Maria Lima Werner holds a D.Sc. in computer science from COPPE/UFRJ (1992). She is an associate professor at COPPE/UFRJ, leads the software engineering group, and is currently responsible for information and communication technology at UFRJ. She is also a CNPq and FAPERJ researcher. She has published several papers and served on program committees of both Brazilian and international conferences and journals. She has extensive experience in computer science with an emphasis on software engineering in the following research areas: software reuse, component-based development, and software product line.