Variability in Software Systems—A Systematic Literature Review

Matthias Galster, Danny Weyns, Dan Tofan, Bartosz Michalik, and Paris Avgeriou

Abstract—Context: Variability (i.e., the ability of software systems or artifacts to be adjusted for different contexts) became a key property of many systems. Objective: We analyze existing research on variability in software systems. We investigate variability handling in major software engineering phases (e.g., requirements engineering, architecting). Method: We performed a systematic literature review. A manual search covered 13 premium software engineering journals and 18 premium conferences, resulting in 15,430 papers searched and 196 papers considered for analysis. To improve reliability and to increase reproducibility, we complemented the manual search with a targeted automated search. Results: Software quality attributes have not received much attention in the context of variability. Variability is studied in all software engineering phases, but testing is underrepresented. Data to motivate the applicability of current approaches are often insufficient; research designs are vaguely described. Conclusions: Based on our findings we propose dimensions of variability in software engineering. This empirically grounded classification provides a step towards a unifying, integrated perspective of variability in software systems, spanning across disparate or loosely coupled research themes in the software engineering community. Finally, we provide recommendations to bridge the gap between research and practice and point to opportunities for future research.

Index Terms—Variability, systematic review, software engineering

1 Introduction

Variability is commonly understood as the ability of a software system or software artifact (e.g., component) to be changed so that it fits a specific context [1]. Variability allows us to adapt the software's structure, behaviour, or underlying processes. These adaptations are enabled through *variation points* and *variants* as options that can be selected at these variation points. Enabling variability in software systems is crucial to ensure that systems successfully adapt to changing needs, and to facilitate the reusability of software systems or individual software artifacts.

So far, variability has mainly been studied in the software product line (SPL) domain [2]. However, variability is a "key fact of most, if not all, systems" [3] and therefore a relevant concern of those systems. Thus, variability is not limited to product lines or families only but imposes challenges on software development in general. Many other types of today's software systems are built with variability in mind, e.g., self-adaptive systems, open platforms, or service-based systems with dynamic runtime composition of services. Situations in which variability must be handled include the configuration of systems, customization of components, dynamic selection of features, and runtime adaptation of a software service. Variability can be facilitated in different ways, e.g., with variant management tools, software configuration wizards and tools, configuration interfaces of software components, and infrastructures for dynamic runtime service composition and adaptation. As variability is pervasive, software engineers need a proper understanding, suitable methods and tools for handling (i.e., representing, managing and reasoning about) variability [4]. This is particularly true if variability needs to be considered by many different stakeholders (e.g., end users who demand variability in the final product, coders who need to be aware where in the code variability is implemented, or testers who need to test all possible variants).

To offer an overview of variability handling in software engineering, this paper reports a systematic literature review (SLR). By variability handling we mean the activity that is part of software engineering to achieve variability in software systems (details are presented in Section 1.1). Our focus is on research studies from premium software engineering journals and conferences. The research method used is based on the guidelines for performing and reporting systematic literature reviews [5]. We deviated from these guidelines in that we did not perform a pure automated search but a targeted manual search that was complemented by a limited and targeted automated search. A justification for this deviation is provided in Section 2. Moreover, we used reports on practical experiences with systematic literature reviews, best practices and lessons learnt (e.g., Staples and Niazi [6], Biolchini et al. [7], Riaz et al. [8], Zhang and Babar [9], Brereton et al. [10]) as well as on meta-studies of systematic reviews (Kitchenham et al. [11], [12]) when designing and conducting our review and when developing our search strategy.

- *M. Galster is with the Department of Computer Science and Software Engineering, University of Canterbury, Private Bag 4800, Christchurch 8140, New Zealand. E-mail: mgalster@ieee.org.*
- *D. Weyns is with the Department of Computer Science, Linnaeus University, Växjö SE-351 95, Sweden. E-mail: danny.weyns@lnu.se.*
- *D. Tofan and P. Avgeriou are with the Department of Mathematics and Computing Science, University of Groningen, Groningen 9700 AK, The Netherlands. E-mail: d.c.tofan@rug.nl, paris@cs.rug.nl.*
- *B. Michalik is with Amartus, Poland. E-mail: bartosz.michalik@gmail.com.*

Manuscript received 2 Dec. 2011; revised 25 Sept. 2012; accepted 24 Nov. 2013; date of publication 11 Dec. 2013; date of current version 28 Mar. 2014. Recommended for acceptance by P. Heymans.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSE.2013.56

0098-5589 © 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

1.1 Variability—An Introduction

Variability is the ability of a software system or software artifact to be extended, customized or configured for (re-) use in a specific context [1]. Thus, variability specifies parts of the software that remain variable and that are not fully defined during early design. This facilitates the development of different versions of a software system or software artifact. Consequently, we interpret variability as planned or anticipated change [13], rather than change due to errors, maintenance or new unanticipated customer needs. This view on variability requires the representation of variation points and variants in software artefacts, throughout the software life cycle [14], e.g., at development time or runtime [15]. Furthermore, variability can support or may be required in functional requirements and/or non-functional requirements (i.e., quality attributes) [16]. According to Svahnberg et al. [17], one reason to support variability is delaying concrete design decisions to the latest point that is economically feasible.

Variability is a key concept in software product lines. Product lines focus on addressing variability explicitly as "features", "decisions" or product configurations. On the other hand, in software engineering in general, variability is often looked at in a broader scope [5] and concerns different stakeholders (e.g., requirements engineers, architects, developers, end users). Moreover, a product line assumes the existence of a product line infrastructure, including related processes (e.g., core asset development, product development, management). This is rarely the case for many software systems which exploit variability.

Throughout this paper we use the term "handling" variability rather than "managing" variability as widely used in the product line community. As argued by Svahnberg et al. [18], managing variability is only one of several activities in the context of variability; managing variability includes the management of dependencies between variations, maintenance and continuous population of variant features with new variants, removing features, the distribution of new variants to the installed customer base, etc. Additional activities, such as identifying variability (i.e., determining where variability is needed), reasoning about, representing and implementing variability (i.e., using a variability realization technique to resolve variability at variation points and to implement a certain variant) exist [18]. We refer to all these activities as "handling" variability.

1.2 Lack of Existing Reviews

To the best of our knowledge, no comprehensive study on research related to variability in software systems exists. Chen et al. [19] reviewed variability management in software product lines. The study found that most current work addresses variability in terms of features, assets or decisions. Most work has been done on variability modeling. Moreover, Chen and Babar [20] assessed the evaluation of variability management approaches in software product lines. Our study differs from these two studies in that we do not focus on the product line domain and variability modeling, but are interested in variability handling in software engineering to achieve variability in software systems in general. Moreover, rather than performing a broad automated search, we perform an extensive manual search on carefully selected software engineering venues, complemented by a targeted automated search.

Kontogogos and Avgeriou [21] reviewed variability in service-based systems. Based on two types of variability (integrated and orthogonal) as defined by Bachmann et al. [22] and Pohl et al. [23], that study identified approaches that apply integrated variability modeling (extending software artifacts with variability) and orthogonal variability modeling (adding new representations of variability separately from existing representations) for service-based systems. The authors found that most current approaches for variability modeling in service-based systems are feature-based and originate from the product line domain. However, the study only identified a small set of approaches, focused on variability in service-based systems and variability modeling. Moreover, the study cannot be considered as a systematic literature review, but as an informal literature survey [11]. Similarly, Kazhamiakin et al. [24] studied adaptation of service-based systems in an informal review.

Furthermore, Alves et al. [25] studied variability in the area of requirements engineering for software product lines. The study aimed at identifying requirements artifacts that current product line approaches deal with, the addressed requirements engineering activities, and product line adoption strategies followed by current approaches. In contrast, our study goes beyond variability in product lines and requirements engineering.

Several other interesting literature reviews in the context of software product lines exist that are related but outside the scope of the goal of our study: Rabiser et al. [26] conducted a systematic literature review on requirements for product derivation support in the context of product line engineering. Benavides et al. [27] reviewed the automated analysis of feature models in product line engineering. Finally, Hubaux et al. [28] investigated the use of feature diagrams in practice.

1.3 Goals and Contributions

Research on variability has mainly been conducted in the domain of product line engineering. A first step towards addressing variability in a more holistic manner is to identify and analyze existing research on handling variability in software engineering, without focusing on any technology domain (e.g., service-oriented architectures, component-based systems), or application domain (e.g., telecommunication, e-commerce, automotive). Therefore, the objective of this study is to summarize existing research related to variability handling in software engineering to support variability in software systems. In particular, we aim a) to appraise evidence of research on variability handling, and b) to identify trends in variability research, open problems and areas for improvement.

The outcome of the systematic review helps identify areas for further investigation of handling variability in specific areas, such as software architectures or service-based systems. Also, lack of evidence could highlight the need for more thorough studies that apply clear research methods; poor evidence could mean that more data about existing methods for handling variability should be collected;

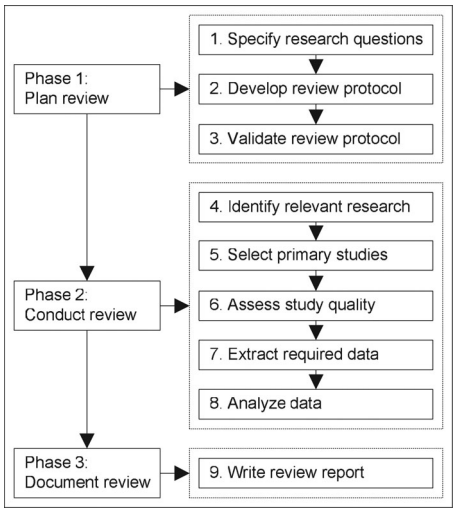

Fig. 1. Systematic literature review process (adapted from [10]).

research trends and open problems could point to exciting research opportunities. Furthermore, the review will help position new research activities: For new research, we need to identify the state-of-the-art and make clear where new research fits into the current body of knowledge. Finally, based on our findings we propose dimensions for variability to provide a more rigorous understanding of variability in software engineering. This classification for variability in different dimensions captures key facets of variability.

1.4 Paper Structure

In Section 2 we discuss the research method. Section 3 presents demographic information and the results we obtained from analyzing the data extracted from reviewed studies. A discussion and interpretation of those results takes place in Section 4. In Section 5, we discuss limitations of our review. Section 6 concludes this paper.

2 Research Method

The systematic literature review method [5] is a well-defined approach to identify, evaluate and interpret all relevant studies regarding a particular research question, topic area or phenomenon of interest [30]. This method was chosen because we wanted to get a fair, credible and unbiased evaluation of approaches for handling variability in software systems. The systematic review was performed as shown in Fig. 1 (adapted from [10]).

An important step is the development of a review protocol to ensure rigor and repeatability. In order to reduce researchers' bias when performing this review, we adapted the process for developing a review protocol proposed by Ali et al. (Fig. 2) [29]. After identifying the research questions, we decided to use a manual search as our search strategy and defined a search scope. As part of this step, we performed pilot searches. Then, we developed study inclusion and exclusion criteria, and defined the search process. Also, we proposed our strategy for assessing the quality of studies that we considered in the review. Next, based on our research questions, we identified what data elements to extract from each study found in the search. A data

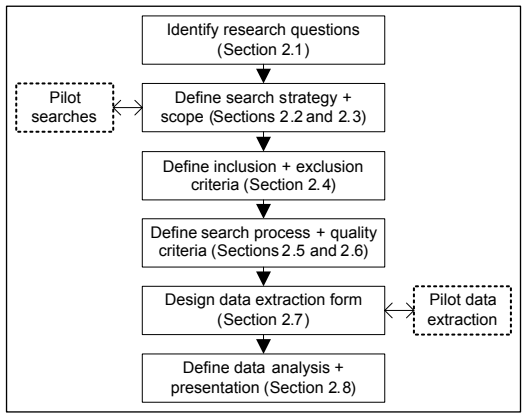

Fig. 2. Process to develop review protocol (adapted from [29]).

collection form was piloted with several papers found in the pilot search. Finally, to crosscheck the selection of the primary studies identified in the manual search we defined search terms for a targeted limited automated search. Also, we defined our strategy to analyze extracted data and to present the results. Details are presented in the remainder of this section.

Five researchers distributed across four locations were involved in the review. Communication took place online and in several on-site meetings. Tools used for the review included EndNote X4, Dropbox and Microsoft Excel.

2.1 Research Questions

We first formulated the review goal through Goal-Question-Metric perspectives (purpose, issue, object, viewpoint) [31]:

Purpose: *analyze and characterize;*
Issue: *handling of variability;*
Object: *in software systems;*
Viewpoint: *from a researcher's point of view.*

Based on the goal, we derived research questions:

RQ1: What methods to handle variability in software systems exist?
RQ1.1: What types of variability do these methods handle?
RQ1.2: What activities in the software development process are affected by these methods?
RQ1.3: What runtime and design time quality attributes are addressed by these methods?
RQ2: What is the evidence that motivates the adoption of existing methods?
RQ3: What are the limitations of the existing methods?

RQ1 is motivated by the need to describe the state-of-the-art of handling variability. RQ1 is refined into three subquestions that aim to answer whether variability occurs and is treated at design time or runtime (i.e., the types of variability), which software development activities are addressed in current research, and what quality attributes (e.g., performance, security) are addressed. We are interested in quality attributes as these are often neglected in the context of variability [32]. Based on Bass et al. [33], we differentiate design time and runtime quality attributes. Examples for design time quality attributes include modifiability or portability. Examples for runtime quality attributes are

availability or performance. RQ1 allows researchers to get an overview of approaches for handling variability. RQ1 relates to the goal of identifying trends in variability research, open problems and areas for improvement, as outlined in Section 1.3.

We formulated RQ2 to find how much evidence is available to apply methods. By evidence we mean any indicator that a method works to address the problem targeted by this method. RQ2 is of interest for researchers to obtain evidence about what methods could be used in practice. Also, it provides researchers with an idea about the maturity of current approaches. RQ2 relates to the goal of appraising evidence of research on variability handling, as outlined in Section 1.3.

As we were interested in identifying gaps in current research, we formulated RQ3. Researchers benefit from answering RQ3 by getting an overview of issues that could provide directions for further research. Thus, RQ3 relates to the goal of identifying trends in variability research, open problems and areas for improvement (Section 1.3).

2.2 Search Strategy

We used a manual search and manually browsed journals and conference proceedings. This manual search was complemented by a targeted automated search to ensure reliability of the studies found in the manual search and to increase reproducibility.

Despite the high effort, a manual search tends to be more thorough than an automated search. A pure automated search using a standard search string on electronic data sources might miss relevant literature. This is particularly relevant for a topic that spans a variety of fields and for which no established terminology exists, such as variability handling. Also, targeted searches at carefully selected venues are justified to omit low quality papers [34]. Grey literature (i.e., published without a peer review process, e.g., reports, theses) that would be identified in an automated search tends to be of low quality [35]. Finally, a manual search allows us to find studies whose authors have not used common terms for variability-related concepts, or did not use keywords that we would have assumed to appear in paper titles, abstracts or keywords [36].

We included any study type (empirical, theoretical, conceptual, etc.). This is because when piloting our study, we could not identify a standard study type in our problem domain. Variability affects software development at various stages. Thus, we organized the search around five general software engineering areas: Requirements engineering (RE), architecture and design (AD), implementation and integration (II), testing and verification (TV), and maintenance (M).

2.3 Scope of Search and Sources Searched

Our search scope and sources are based on:

1. Preliminary searches for existing reviews (e.g., Chen et al. [19], [37], Williams and Carver [13]);
2. Pilot searches based on the research questions;
3. Reviews of research results (e.g., papers at the Software Product Line Conference (SPLC), or other resources, e.g., www.variabilitymodeling.org);
4. Consultation with other researchers in the field.

The scope of our search is defined in the three dimensions of time, space, and quality:

1. Publication period (time): We searched papers published between July 2000 and (including) 2011. July 2000 was chosen because the first SPLC (the premium venue for variability and variability management) was held in 2000 (before, only informal workshops were held). Starting the review in 2000 guarantees a certain maturity of studies.
2. Publication venue (space): As we performed a manual search, we applied rigorous selection criteria for venues to be searched. First, we included SPLC and GPCE, two premium venues where research related to variability is published. Second, we included venues that are known for publishing high quality software engineering research in general (GSE). Third, we chose the premium venue for each of the five software engineering areas listed in Section 2.2.
3. Venue quality: To ensure a level of quality of papers, we used two selection criteria. First, we only included venues which are evaluated by the Australian Research Council higher than or equal to level "B" [38]. We include "B" as for some software engineering areas it was not possible to identify an "A" venue. Furthermore, rankings of scientific venues are usually not conclusive and vary between ranking systems. Second, we used the H-index of venues as a criterion for venue quality[1] and included venues with an H-index of at least 10. However, we included EASE, ECOOP, FSE, SLE and ECSA, which have no or a lower H-index because they are considered important venues in the respective communities. Searched venues had to be strictly from the software engineering domain. Other venues of very high quality and with a high ranking and a large H-index (e.g., Communications of the ACM) were not included since they target a diverse audience and therefore typically do not present in-depth research studies on specific topics, such as variability.

The selected venues are listed in Table 1 (13 journals) and Table 2 (18 conferences). The tables also show the software engineering areas outlined in Section 2.2. For the manual search we did not run search queries on electronic search and indexing machines, such as IEEE Xplore or ACM Digital Library. Instead, we used these electronic resources for targeted searches by manually browsing through tables of contents (including keywords and abstracts of papers, and full papers) of journals and conference proceedings available through IEEE Xplore, ACM Digital Library, etc. As venues are potentially indexed by different sources, we provide an overview of the sources we searched for each venue in Table 12 in the appendix. Some conferences have more than one source as publishers of proceedings changed over years.

We complemented the manual search with a limited targeted automated search on Scopus. The automated

1. The H-index for journals is based on http://www.scimagojr.com, for conferences we used http://academic.research.microsoft.com/; last access: May 1, 2013.

TABLE 1
Searched Journals

ID	Area	Journal	Rank	H-index
ASEJ	GSE	Automated Software Engineering Journal	A	23
ESE	GSE	Empirical Software Engineering: An International Journal	A	28
IEEE SW	GSE	IEEE Software	B	57
IST	GSE	Information and Software Technology	B	37
JSS	GSE	Journal of Systems and Software	A	43
REJ	RE	Requirements Engineering Journal	A	23
SCP	GSE	Science of Computer Programming	A	35
SMRP	M	Software Maintenance and Evolution: Research and Practice	B	21
SoSyM	GSE	Software and Systems Modeling	B	19
SPE	GSE	Software: Practice and Experience	A	41
STVR	TV	Software Testing, Verification and Reliability	B	25
TOSEM	GSE	ACM Transactions on Software Engineering Methodology	A*	44
TSE	GSE	IEEE Transactions on Software Engineering	A*	88

TABLE 2
Searched Conferences

ID	Area	Conference	Rank	H-index
ASE	GSE	Automated Software Engineering Conference	A	44
CAiSE	GSE	International Conference on Advanced Information Systems Engineering	A	29
EASE	GSE	Evaluation and Assessment in Software Engineering	A	n/a
ECOOP	II	European Conference on Object-oriented Programming	A	n/a
ECSA	AD	European Conference on Software Architecture	n/a	8
FSE[2]	GSE	Foundations of Software Engineering	A	n/a
GPCE	GSE	Generative Programming and Component Engineering	B	27
ICSE	GSE	International Conference on Software Engineering	A	118
ICSM	M	International Conference on Software Maintenance	A	57
ICST	TV	International Conference on Software Testing, Verification and Validation	C	12
ISSTA	TV	International Symposium on Software Testing and Analysis	A	35
MODELS[3]	GSE	International Conference on Model Driven Engineering Languages and Systems	B	24
OOPSLA	II	Object-Oriented Programming, Systems, Languages, and Applications	A	59
QoSA	AD	Conference on the Quality of Software Architectures	A	10
RE	RE	International Requirements Engineering Conference	A	47
SLE[4]	GSE	International Conference on Software Language Engineering	B	n/a
SPLC	GSE	International Product Line Conference	n/a	28
WICSA	AD	Working Conference on Software Architecture	A	25

[2] *FSE includes the European Software Engineering Conference (ESEC) for years in which FSE and ESEC were co-located.*
[3] *Before 2005, MODELS was the International Conference on the Unified Modeling Language, Modeling Languages and Applications (UML).*
[4] *First edition of SLE took place in 2008.*

search was targeted and limited in that it covered the same time period as the manual search and venues included in the manual search. We used keywords and their variations (e.g., plural) that are common in our domain of interest: "variability", "variation point", "variant", "feature model", "feature diagram", "product family" and "product line" (complete search string can be found in [39]). Furthermore, we searched each conference individually using the search engines available from the databases listed for each conference in Table 12 and applying the same time range and keywords as for the search on Scopus. We did this because the description of Scopus with regard to the inclusion of all editions of all conferences in Table 12 is not clear. The automated search resulted in 1,503 papers out of which we identified 91 papers also in the manual search (for example, we found several of the papers related to software product lines such as studies 20 and 146 in the list of papers included in the review as shown in Table 15 in the appendix). Furthermore, the automated search found 1,412 papers that we had not selected in our manual search. They were not included in our review because they did not meet the inclusion criteria or met exclusion criteria (e.g., preface to the special issue on software evolution, adaptability and variability in SCP, or papers about software failure prediction or investigating source code, without focusing on variability). Finally, the automated search also found 28 papers that were relevant to our study but which we missed in our manual search and thus were added to the manual search results and included in the 91 papers found in both searches (e.g., study 193, 195, 196). Thus, the automated search helped us get a more complete set of studies for our review. However, the manual search found 105 studies whose authors have not used common terms for variability-related concepts, or did not use keywords that we would have assumed in paper titles, abstracts (e.g., study 1). This confirms our initial assumption that a pure automated search using a standard search string on electronic data source would miss relevant literature.

When searching journals, we ignored calls for papers or participation, society or journal information, covers, etc. We excluded tutorial or workshop reports, poster sessions and companions when searching conferences.

2.4 Inclusion and Exclusion Criteria

Inclusion and exclusion criteria helped identify studies directly related to our research questions. We are interested in empirical studies, but also in studies that propose new concepts and use empirical methods for evaluation. By having strict search venue selection criteria, we had already applied some inclusion criteria before the initial manual

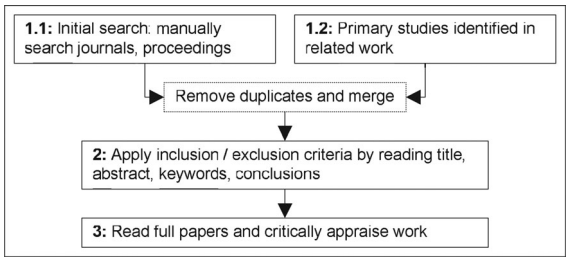

Fig. 3. Search process.

search at Stage 1.1 of the search process (Fig. 3), e.g., excluded studies that were not from an acceptable source). Inclusion and exclusion criteria were applied during the initial search as well as later to studies that had been identified through the initial search. A paper was selected as a primary study if it met all inclusion criteria and eliminated if it met any exclusion criterion:

- *Inclusion criterion 1*: Study is internal to software domain. We are only interested in variability in the software domain. Studies about hardware or other types of systems are not relevant.
- *Inclusion criterion 2*: Study comes from an acceptable source. We aim at gaining solid information about variability and thus ignored lower-quality sources. This criterion was implicitly met by carefully choosing the search venues.
- *Inclusion criterion 3:* Study is about extending, changing or customizing software systems or software artifacts.
- *Inclusion criterion 4:* Study is about reusing software systems or software artifacts in different contexts.
- *Inclusion criterion 5:* Study discusses how to design parts of the software that remain variable and are not fully defined during early design.
- *Inclusion criterion 6:* Study is about planned/anticipated change/reuse, rather than about unforeseen change.
- *Exclusion criterion 1*: Study is an editorial, abstract, position paper, short paper, tool paper, poster summary, keynote, opinion, tutorial summary, conference summary (or introduction to conference proceedings), workshop summary or panel summary. Editorials, abstracts, etc., do not provide a reasonable amount of information.
- *Exclusion criterion 2*: Study is about general software maintenance or evolution independent of variability (such as corrective change or bug fixing). We are only interested in planned adaptation.

2.5 Search Process

We used a staged process (Fig. 3). At Stage 1.1, each journal and conference venue was manually reviewed by two researchers, who read title, keywords, and abstract to determine a paper's relevance. Differences were reconciled collaboratively. Stage 1.2 searched studies identified in related reviews, namely [19], and followed up on references in studies found at Stage 1.1. Duplicates from Stage 1.1 and Stage 1.2 were removed and results merged.

TABLE 3
Questions to Assess Study Quality

#	Question
Q1	Is there a rationale provided for why the study was undertaken?
Q2	Is there an adequate description of the context (industry, laboratory setting, products used, etc.) in which the research was carried out?
Q3	Is there a justification and description for the research design?
Q4	Is there a clear statement of findings, including data that supports findings?
Q5	Did the researcher(s) critically examine his / her (their) own role, potential bias, and influence during the study?
Q6	Are limitations and credibility of the study discussed explicitly?

At Stage 1.1, if there was any doubt whether a study should be included, it was added to the list of potentially relevant studies. Abstracts might be insufficient to rely on when selecting studies [10]. Thus, at Stage 2 if necessary, we also decided about study inclusion based on the conclusions of studies. Full copies of studies were obtained for remaining studies (i.e., studies left after filtering at each stage). Final inclusion/exclusion decisions were made after full texts had been retrieved (Stage 3). In case of multiple studies referring to the same method, only the most recent was included. After each stage, one researcher randomly distributed the remaining studies between the reviewers. "Randomly" means that we did not ask reviewers to pick their favorite papers, but simply created batches of papers in the order they occurred in the reference manager tool. The batches given to reviewers were the same, but different reviewers may have reviewed different batches (i.e., not the same two reviewers always reviewed exactly the same batches, but one pair of reviewer reviewed the same batch).

2.6 Quality Criteria

For systematic literature reviews it is critical to assess the quality of primary studies [5]. Thus, all studies were assessed through a quality check. As proposed in [40], papers were assessed using questions (Table 3). We preferred this approach instead of using the study design hierarchy for software engineering proposed in [30]. This is because a study design hierarchy classifies empirical studies. However, as previous reviews have shown (e.g., [20]), we could expect to find many empirical studies. This means, elements of a study design hierarchy (e.g., experimental studies, case control studies) would not apply to many studies.

Similar as Ali et al. [29], we adopted the quality assessment instrument used in [41]. This instrument uses a three-point scale to answer each question, either as "yes", "to some extent" or "no". By including "to some extent" we avoided neglecting statements where authors provided only limited information to answer the assessment questions [29]. Each quality assessment question was answered by assigning a numerical value (1 = "yes", 0 = "no", and 0.5 = "to some extent"). Then, a quality score was given to a study by summing up the scores for all questions for a study [29]. As can be seen in Table 3, these scores refer to the quality of the reporting of a study, rather than its actual quality. Note that quality criteria were not used for filtering purposes to include or exclude primary studies in the review

TABLE 4
Data Collection Form

#	Field	Research question
F1	Author(s)	n/a
F2	Year	n/a
F3	Title	n/a
F4	Venue (journal or conference)	n/a
F5	Keywords	n/a
F6	Abstract	n/a
F7	Citation count	RQ2
F8	Quality score	RQ2
F9	Runtime quality attributes	RQ1.3
F10	Design time quality attributes	RQ1.3
F11	Level of tool support	RQ3
F12	Evidence level	RQ2
F13	Activities addressed by approach	RQ1.2
F14	Limitations	RQ3
F15	Types of variability	RQ1.1
F16	Type of study	RQ2

but to assess the quality of all included studies. Furthermore, the quality criteria were applied to all studies included in the review, rather than only to papers that report empirical studies.

2.7 Data Collection

For each study, the information shown in Table 4 was collected. Studies were read in detail to extract the data. Three researchers read the selected studies in parallel. For each paper, data was extracted by one researcher and checked against the paper by another researcher. Disagreements were resolved by discussions between researchers or by consulting an additional researcher.

Data for F1 to F6 were collected for documentation purposes to keep meta-information of papers. F7 records the citation count for each paper, based on Google Scholar (as of May 2013). F8 records the quality score for each paper, based on criteria introduced in Section 2.6. F9 and F10 record quality attributes. We recorded quality attributes that are addressed in studies, or by proposed techniques. F11 records if there is any tool support. Tool support could be automatic, semi-automatic or manual. The evidence level (F12) of a study is critical for researchers to identify new topics for empirical studies, and for practitioners to assess the maturity of a particular method or tool. Kitchenham proposed five levels of study design [30]. We used a revised classification to make the assessment more practical [25]. From weakest to strongest, the classification is as follows: 0 = no evidence; 1 = evidence from demonstration or toy examples; 2 = evidence from expert opinions or observations; 3 = evidence from academic studies (e.g., controlled lab experiments); 4 = evidence from industrial studies (e.g., causal case studies); 5 = evidence from industrial practice. According to Alves et al. [25], industrial practice indicates that a method has already been approved and adopted by industrial organizations. Thus, practice shows convincing evidence that something works and is thus ranked strongest. Activities addressed (F13) are requirements engineering, architecture/design, implementation, testing and verification, and maintenance. Limitations (F14) and types of variability (F15) were recorded as text. Types of variability are dynamic variability (i.e., variability resolved at runtime) and static variability (i.e., variability resolved during design/development time). F16 records if a study is an empirical paper or not.

2.8 Data Analysis

Data from studies were collated to answer the research questions. Most papers in our review are grounded in qualitative research. As argued by Dyba et al. [42], meta-analysis might not be suitable for synthesizing qualitative data. Thus, the data were manually reviewed in the spreadsheets where they were stored. As found by other researchers, tabulating the data was useful during aggregation [10]. On parts of the data we used descriptive statistics and frequency analysis. Frequency analysis has been used by systematic reviews which deal with qualitative data [20].

2.9 Protocol Review

The protocol was reviewed by all researchers involved in the review as well as by external researchers not involved in the review. Revisions to the protocol were made accordingly (e.g., clarification of research questions and the data collection form, refinement of search venues). Moreover, the systematic review was validated as follows:

1. Researchers used a subset of resources to test the search process. Problems in replicating the process were identified and the process revised.
2. Gaps in search venues were identified and venues changed to include relevant venues.
3. The reliability of data extraction was tested. A researcher was given a set of papers and asked to fill in the data extraction form. Based on the feedback revisions were made to the protocol.

During the review we deviated from the original protocol at some points. Initially we planned to record if papers include a discussion on impact assessment of variability, or if there is any sort of feature handling used. However, it turned out that this information was difficult to obtain. Thus, the corresponding data fields were removed from the data collection form.

3 Results

3.1 Demographic Data

We manually searched 15,430 papers. We looked at 7,991 journal papers (Table 13 in appendix) and 7,439 conference papers (Table 14 in appendix). The number of papers after Stage 2 was 390 (168 journal papers + 222 conference papers). Twenty-eight papers also resulted from the targeted automated search. During data collection, more papers were removed due to their lack of relevance for the study (see also Section 2.5). This resulted in 196 papers for the final review (see Table 15 in the appendix). Fig. 4 shows the number of papers included in the review, per journal. Fig. 5 shows the number of papers per conference.

We compared our primary studies with related reviews listed in Section 1.2. We noticed only a small overlap between our primary studies and primary studies included in other reviews. For example, Chen and

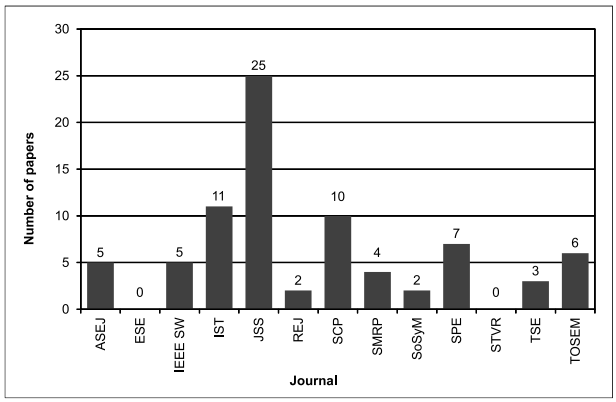

Fig. 4. Number of papers per journal.

Babar [20] reviewed 97 papers. Only 14 papers are also included in our review. The complete comparison is available as supplementary material in a technical report [39]. The limited overlap is due to the different goals and foci of the studies. On the other hand, we conclude that our extended scope (beyond product lines) is indeed reflected in the studies included in our final review.

JSS has been the most prominent journal to publish variability-related research, whereas SPLC is the most prominent conference for variability research. Searching the two venues that focus on empirical research (EASE and ESE) did not result in a significant number of studies. This confirms other reviews that also could not identify a study on variability modeling published in ESE [20]. Furthermore, none of the testing and verification venues (ICST, ISSTA, STVR) or development-related conferences (ECOOP, OOPSLA) seem to be attractive venues for variability research.

Fig. 6 shows the distribution of studies based on the venues related to the software engineering areas introduced in Section 2.2. Fig. 6 shows that most studies have been published in general software engineering venues.

Finally, Fig. 7 shows the distribution of studies over the years from 2000 to the end of 2011. This figure shows an increasing trend for publishing variability research until 2008. After 2008, the increase in publications seems to slow down. One factor that explains this trend could be the emerging interest in the use of variability in dynamic service composition.

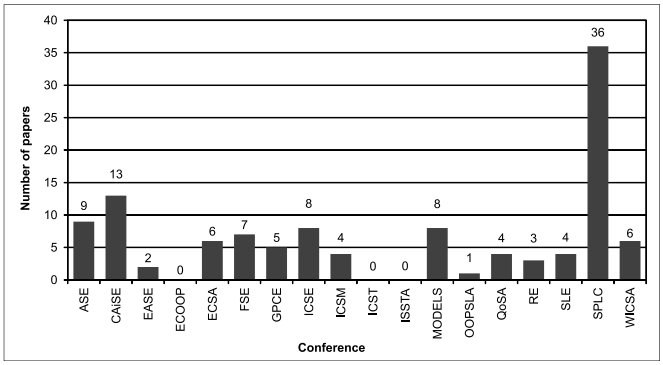

Fig. 5. Number of papers per conference.

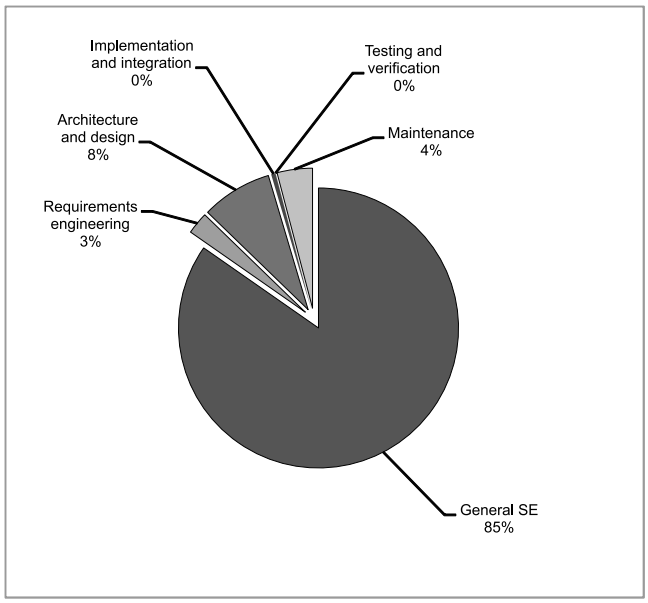

Fig. 6. Studies per software engineering area of publication venue.

3.2 RQ1: What Methods to Handle Variability in Software Systems Exist?

We now discuss studies on variability in software systems (studies are listed in the appendix, Table 15).

3.2.1 *RQ1.1: What Types of Variability Do These Methods Handle?*

To answer this question, we drew on data extracted based on F15 (types of variability) from the data extraction form. Overall, 128 studies focus on design time variability and 50 studies on runtime variability. The remaining 18 studies do not clearly differentiate between runtime and design time variability.

Observations. Design time variability can be considered the current trend in variability research. However, recent trends, such as self-adaptive systems [43], runtime orchestration of service-based systems [44] and the emerging

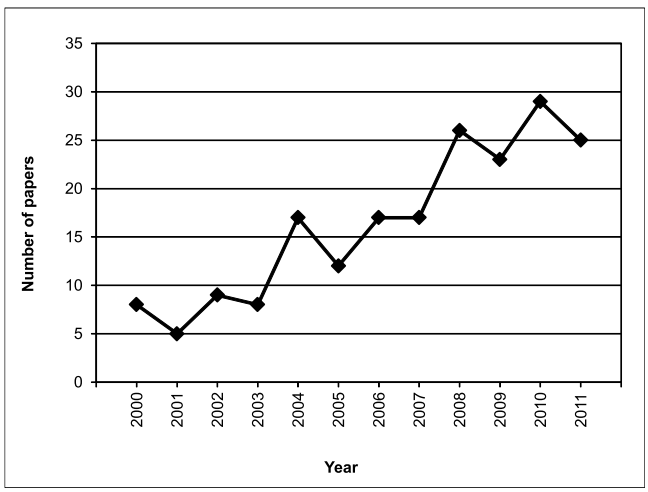

Fig. 7. Studies per year.

TABLE 5
Software Engineering Activities Addressed by Studies

	Absolute	%
Requirement engineering	32	13.0
Architecture	72	29.2
Design	86	34.8
Implementation	25	10.1
Testing and verification	5	2.0
Maintenance	27	10.9

domain of dynamic product lines [45] have led to increasing attention to runtime variability.

3.2.2 RQ1.2: What Activities in the Software Development Process Are Affected by These Methods?

To answer this question, we used data extracted based on F13 (activities addressed by approach) from the data extraction form. We were interested in variability that could occur anywhere in the software development cycle. Development activities addressed are similar to the software engineering areas used for selecting search venues. Therefore, the activities used during data analysis were requirements engineering, architecture, design, implementation, testing and verification, and maintenance. We found that some studies did not explicitly address any of these activities, whereas other studies addressed more than one activity. In cases where more activities were addressed, we assigned studies to all activities they address. Thus, the sum of the second column in Table 5 exceeds the number of studies included in the review (i.e., 196).

Observations. Overall, 130 studies only address one single activity. Table 5 shows that most studies deal with variability during the architecture and design phase. Additionally, we found that 22 studies address both activities, architecture and design (studies 39, 59, 61, 62, 65, 76, 80, 136, 138, 143, 146, 167, 169, 170, 171, 172, 173, 174, 187, 189, 191, 193). Only one study (study 51) addresses six activities (all except design).

3.2.3 RQ1.3: What Runtime and Design Time Quality Attributes Are Addressed by These Methods?

To answer RQ1.3, we used data extracted based on F9 (runtime quality attributes studied) and F10 (design time quality attributes studied). To collect all possible quality attributes and to be not limited to a fixed list, we did not apply a predefined list of quality attributes. Instead, the list of quality attributes emerged from the quality attributes extracted from the studies. Runtime quality attributes that have been mentioned in studies are:

- Availability (20 studies): Ability to be operational and accessible when required for use [46].
- Evolvability (seven studies) and flexibility (21 studies): Ability of a system to accommodate changes in requirements throughout lifespan with low cost while maintaining architectural integrity.
- Interoperability (four studies): Ability to interact with one or more specified system [47].
- Performance (24 studies): Ability to accomplish designated functions within given constraints in terms of speed, accuracy, or memory usage [46].
- Reliability (three studies): Ability to perform required functions under stated conditions for a specified period of time [46].
- Scalability (three studies): Ability of handling increase in computing capacity [48].
- Security (two studies): Ability of system to resist unauthorized usage while still providing its service to legitimate users [33].
- Safety (one study): Ability to avoid catastrophic consequences on the users and the environment [48].

Seven studies mention quality of service (QoS) in the context of variability, without further specifying what quality attribute QoS exactly refers to. Furthermore, some studies do not mention any runtime quality attribute at all. Similarly, most studies do not explicitly discuss design time quality attributes. Design time quality attributes addressed by studies include the following:

- Evolvability (four studies)
- Flexibility (13 studies)
- Modifiability (including reconfigurability and maintainability; four studies)
- Portability (two studies)
- Reusability (six studies)

Observations. We could not identify sets of commonly addressed runtime or design time quality attributes. Runtime quality attributes are addressed more extensively than design time quality attributes. However, our results indicate that handling variability in quality attributes is usually not the focus of current approaches, i.e., variability in quality attributes is usually not expressed explicitly. A reason for this is that many papers stem from the product line domain which traditionally focuses on variability in features (with features understood as sets of functionality).

3.3 RQ2: What Is the Evidence that Motivates the Adoption of Existing Methods?

Evidence to adopt proposed methods is derived from two types of information: The quality scores (F8) and the evidence level (F12). Furthermore, we used the type of study (F16) to investigate if there is a difference between empirical and non-empirical papers. Finally, we used the citation count to identify if there is any correlation between evidence and citations of a paper (F7).

Fig. 8 shows a frequency analysis of the scores for each quality question in Table 3. Furthermore, Fig. 8 differentiates scores for empirical and non-empirical studies. Most studies provide a rationale for why the proposed work would be needed (Q1). Also, most studies describe (at least partially) the context in which they have been conducted (Q2). However, most studies do not properly describe a research design (Q3) and fail to critically examine the role of researchers involved in the study (Q5). Most studies present only partial data that would support findings from a study (Q4). Similarly, most studies present only a limited discussion of their credibility (Q6). To identify whether there is a difference between empirical papers and non-empirical papers with regard to quality scores, we performed a

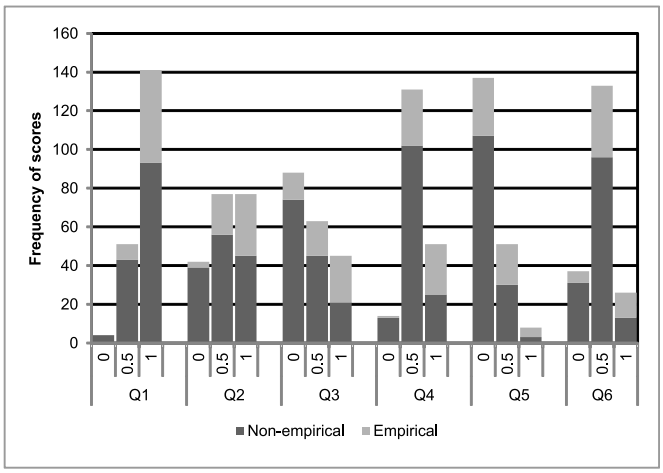

Fig. 8. Frequency analysis of quality scores for each quality question.

Mann-Whitney U test on two independent groups (empirical papers, non-empirical papers). The test showed a statistically significant difference with regard to all six quality questions. In general, empirical studies achieved higher scores across all questions. This observation confirms that empirical studies are conducted and reported in a more systematic and strict manner. However, it is interesting that even the minority of empirical papers critically examines the role of researchers (Q5).

Fig. 9 shows the distribution of total quality scores. The maximum total score is 6 (if a study received a score of 1 for every quality question). Most studies received a score between 1.5 and 3.5. This means, on average the quality of the studies is neither perfect, nor does it suggest that studies are completely flawed. In Table 6 we list the studies with the highest quality scores (i.e., a quality score of 6 and 5.5).

In Table 7 we list the studies with at least 100 citations. We list studies with the highest quality scores and a citation counts because these studies are more likely to be used by researchers and practitioners. A citation count of 100 is chosen because this is the citation count of the top 15 percent papers. However, these numbers are only used as an arbitrary threshold to point out high-quality studies. Table 7 also includes the number of average citations until 2013, based on the publication year. There is no correlation between the average citation count and the total quality score for all 196 papers (Pearson's correlation coefficient $r = 0.1167$; $r = 0.0396$ if correlation between total citation count and total quality score is computed).

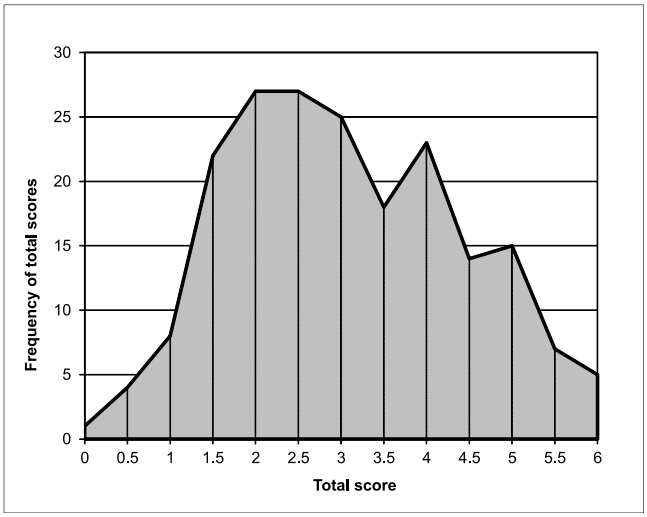

Fig. 9. Distribution of total quality scores.

TABLE 6
Studies with a Total Quality Score of 6 and 5.5

Study ID	Quality score
5	6
16	6
42	6
46	6
57	6
17	5.5
52	5.5
59	5.5
66	5.5
144	5.5
146	5.5
187	5.5

The distribution of evidence levels is shown in Fig. 10 (according to evidence levels introduced in Section 2.7). This figure shows that most studies have a low evidence level and evidence has only been obtained from demonstrations or toy examples. On the other hand, there is no study that shows industrial evidence. Furthermore, Fig. 10 shows

TABLE 7
Studies with Citation Count $>$ 100

Study ID	Citation count	Average citation count	Total quality score
5	428	71	6.0
7	198	15	1.0
11	130	12	4.5
14	168	17	3.5
17	229	25	5.5
20	126	14	2.5
29	173	35	2.5
33	491	49	3.5
35	192	21	2.0
38	197	20	5.0
45	107	13	3.5
50	150	30	2.5
54	202	29	4.5
75	114	19	4.0
76	193	21	3.0
81	128	11	4.5
85	108	12	4.0
96	116	11	3.0
97	124	11	3.0
113	171	21	4.0
124	218	44	3.5
134	101	34	2.5
135	197	20	1.5
145	380	48	1.5
170	345	43	2.0
173	681	52	2.5
185	167	28	2.0
192	225	25	1.5
194	106	35	4.5

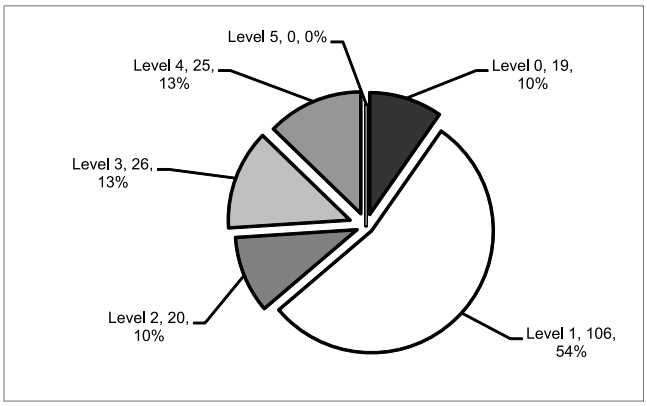

Fig. 10. Distribution of evidence levels.

that many studies have the same evidence level (evidence level 1) and thus cannot be differentiated based on the evidence level. Finally, we list studies with the highest evidence level in Table 8. Again, these studies are potentially most relevant for researchers and practitioners. Pearson's correlation coefficients for evidence level versus both the average citation count and the total citation count are close to zero. Fifty-five papers are empirical papers (i.e., include an empirical study, e.g., as part of evaluating a method). The average quality score of empirical papers is 3.8. The average quality score of other papers is 2.8. Similar as with quality scores, we found no correlation between the average citation count and the evidence levels of all papers (Pearson's correlation coefficient $r = 0.0344$; $r = -0.0084$ for correlation between total citation count and evidence levels).

Observations. Authors provide a description of the motivation and context for the research, but research design, data to support findings, and critical reflection are often insufficiently described. The total quality scores for the studies are moderate, and although these numbers refer to the way studies are reported, it is difficult to judge the quality of a study if it is not reported in detail. Finally, most studies have a low evidence level. Several researchers in the wider software engineering community have repeatedly mentioned the need for systematic evaluation, e.g., [49], [50]. Our study confirms that this observation also applies to the variability handling in software engineering.

3.4 RQ3: What Are the Limitations of Existing Methods?

To answer this question, we used field F14 of the data collection form. Also, we analyzed data about tool support provided in studies (F11); lack of tool support would be a limitation for the practical applicability of an approach. Based on the data from F14, we identified the following recurring themes in reported limitations:

- Correctness and consistency are difficult to guarantee; authors report the need for verification, consistency checking, etc.
- Limitations regarding quality attributes, in particular performance overhead, lack of support for security needs and dynamic features, such as runtime upgradability.

TABLE 8
Studies with Highest Evidence Level

Study ID's	Evidence level
3, 6, 17, 19, 46, 51, 57, 74, 87, 92, 97, 113, 116, 117, 118, 122, 126, 128, 129, 132, 146, 151, 165, 166, 195	4

- Poor user-friendliness, e.g., in terms of flexibility and user support for variability handling.
- Limitations regarding the identification of variability and variants, e.g., some authors report that more advanced techniques such as data mining could be applied.

Other limitations are related to specific issues, such as notations or models to be used. With regard to tool support (F11) we found that 104 studies include some sort of tool support. Sixty-seven studies do not provide tool support and the remaining studies do not discuss any tool support. Other limitations related to weak validation or study quality have been discussed in Section 3.3.

Observations. The reported data confirm several findings we already discussed, in particular, limited coverage of quality attributes, poor validation, and the need for better tool support. In addition, two identified tracks of research that is required are formal methods to provide guarantees about variability (consistency, correctness), and disciplined approaches for composition (variant binding), in particular runtime composition. While additional costs are also reported as a limitation, most studies do not provide details on the exact cost of the proposed approaches, let alone a cost-benefit analysis.

4 Discussion of Results

4.1 Application of Methods to Handle Variability in Software Engineering

The results to RQ1 in Section 3.2 present the state-of-the-art methods to handle variability. A further analysis of papers showed that 46 papers are related to SOA. These papers include 27 papers that are about runtime variability. The data are available as supplementary material in a technical report [39]. This could imply that there is a strong belief in the research community that SOA is one way of handling variability. On the other hand, SOA in general seems to be a popular research area in the period of our search that started in the year 2000. However, even though SOA provides some "built-in" flexibility by dynamic service binding, a pure service-based solution is not enough to thoroughly address variability. For example, service-based systems are changed at runtime [51]. Consequently, to fully support variability in service-based systems, events that occur in such systems must be linked to rules to reason about variants. Also, as software services may be consumed by anonymous consumers, it becomes very difficult, if not impossible, to have a centralized authority that handles variability in the individual parts of the system. Moreover, services are developed and deployed independently in a networked environment and developers integrate services, third-party applications, organization-specific systems and legacy systems. Thus, coordinating variability between different

instances of a system becomes difficult [60]. This is particularly true when considering the results in Section 3.2.2, which suggest that many activities are affected by variability handling. Additional research is required to tackle the challenges of variability handling in open service environments.

Furthermore, our results indicate that a fair share of studies concerns the activities of requirements, implementation and maintenance. This confirms our statement from the introduction that variability is a pervasive topic, relevant throughout the software engineering cycle. We only found one activity, testing, where variability has been insufficiently addressed. Thus, investigating testing in the context of variability is an interesting topic for future research. Some initial work on testing in the context of variability has been proposed in the product line domain [52], [53].

With regard to quality attributes we found that performance is addressed most, whereas quality attributes such as security or safety are rarely a concern of variability handling approaches. Overall, quality attributes still play a minor role in the context of handling variability. This is surprising because the importance of quality attributes has been acknowledged in the software architecture domain. Consequently, studying the relationship between quality attributes and variability is a promising area for future work.

Another observation is that no particular technology receives special attention in current work. This has the benefit of current work being potentially widely applicable. On the other hand, specifics of certain domains and in particular complex domains (e.g., telecommunication, medical devices, etc.) often require domain-specific solutions. We consider the extension of variability approaches for specific domains as a promising direction for future work.

4.2 Applicability of Results in Practice

The results of RQ2 in Section 3.3 present available evidence for the validity of approaches for variability handling in software engineering. We obtained these findings from the premium software engineering venues. However, the moderate quality of how studies are reported (quality questions) as well as the low evidence levels imply that we should carefully interpret the validity of approaches reported. On the other hand, the lack of any sort of tool support (research prototypes, demos, plug-ins for commercial tools, etc.) makes it difficult for practitioners to use or at least explore any existing method and thus inhibits the transfer of results from research into software engineering practice. Researchers should provide more tools in order to make the use of new approaches attractive for practitioners.

Another issue is that proposed methods have been applied to small systems but not to industry-scale projects. A concern of practitioners could be how approaches would affect development activities and where to "plug in" variability handling. We have shown in Section 3.2.2 that most approaches relate to architecture and/or design. However, we did not find clear information about the real overhead involved in using systematic approaches for handling variability, compared to ad-hoc variability handling.

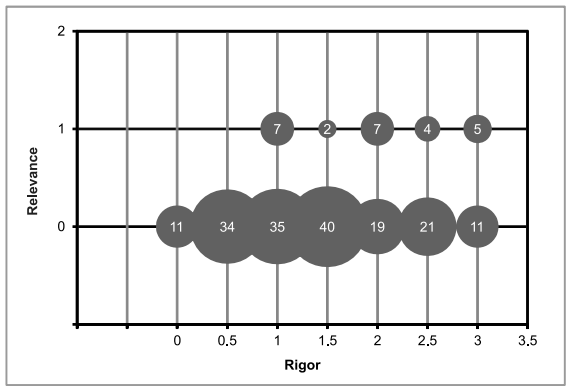

Fig. 11. Relevance versus rigor.

To further analyze the applicability of the reported methods for variability handing in practice, we evaluated the relevance and rigor of current research. Relevance refers to the potential impact of research on academia and industry. To evaluate relevance, Ivarsson and Gorschek use the realism of the environment in which studies are conducted (e.g., the scale and context of a study) [54]. Thus, we used evidence levels to evaluate relevance. As recommended in [54], relevance is scored binary; that is, evidence level 4 or 5 are scored 1 for relevance, whereas evidence levels 1 to 3 are scored 0 [55].

Rigor is about how an evaluation is performed and how it is reported. If a study is poorly reported, rigor of the evaluation cannot be evaluated properly [54]. Ivarsson and Gorschek propose a score for rigor based on the extent to which context, study design, and validity are described [54]. We mapped quality questions (Table 3) to these factors as follows: "Q2: Is there an adequate description of the context in which the research was carried out?" was mapped to the "context" aspect, "Q3: Is there a justification and description for the research design?" was mapped to "study design", and "Q6: Are limitations and credibility of the study discussed explicitly?" was mapped to "validity". We used a similar numerical scoring system as introduced in Section 2.6 to calculate the rigor value (1 = "strong", 0 = "weak", and 0.5 = "medium"). A total rigor score was given to each study by summing up the scores for all the questions (i.e., $\text{rigor} = \text{Q2} + \text{Q3} + \text{Q6}$). Fig. 11 shows the results for rigor and relevance of the studies. The size of the bubbles in Fig. 11 reflects the number of papers.

The five papers with highest rigor and relevance are papers 17, 46, 51, 57 and 146. Three of these papers (17, 46, 146) address the design activity, architecting (51, 57, 146) and implementation (17, 46, 51). Testing is addressed by one paper (51) and requirements engineering by two papers (46, 51). Similarly, the papers with lowest rigor and relevance are papers 10, 111, 138, 178, 181, 182, 183, 194, 186, 190 and 193. All these papers address different development activities. Thus, relevance and rigor of studies do not depend on the development activity.

Overall, the center of gravity in Fig. 11 shows that relevance and rigor of the research studies is rather weak. This confirms our finding of RQ2 that some reservation is recommended when interpreting the validity of reported approaches regarding applicability in practice.

4.3 Dimensions of Variability

The primary studies from the review offer a basis to provide a more rigorous and holistic characterization of variability in software systems, beyond the mere notion of variability as the ability to change or customize a system. In this section, we present a classification of variability in different dimensions that capture key facets of variability. The goal of the classification is to establish a baseline from which key aspects of variability of different types of software systems can be identified and compared. It is not our ambition to be conclusive, nor do we claim that this is the only possible classification of variability in software systems. However, we believe that the empirically grounded set of core dimensions of the classification offers a basis towards defining a comprehensive classification of key properties of variability in software systems.

4.3.1 Definition of Dimensions

To define the dimensions, we started from an initial set of candidate dimensions that we derived from insights gained from the literature review and related studies that classify aspects of variability and adaptability, including [18], [24], [56]. The identified dimensions concern both the description of variability and the realization and effects of variant binding. The initial set consisted of 17 dimensions.[5] Some of the identified dimensions related directly to data collected during the literature review, others related only indirectly. For example, based on runtime quality attributes (F9) and design time quality attributes (F10), we identified requirement type as a dimension. Similarly, we identified automation based on tool support (F11). For each dimension, we identified a range of possible options (for example, dimension requirement type has the options "functional" and "quality"). Then, we performed a pilot study in which we applied the candidate dimensions to a representative set of primary studies of this literature review: For each of the activities addressed by the studies (F14) we analyzed the top 20 percent of studies with the highest quality score (F8). This selection guarantees that we cover studies with the best quality of reporting in the different phases of the software development cycle. Each study was analyzed independently by two reviewers and conflicts were resolved in discussion with at least one extra reviewer.

During the initial phase of the pilot (in two iterations, after collecting data for four and eight studies), we revised the dimensions by removing several dimensions because they are domain-specific rather than variability-specific, including predictability, criticality and frequency of binding; openness was removed as it was meant to describe to what extent systems with variability are constrained, but differentiating between degrees of "constrainedness" is rather subjective; time to bind (i.e., how long does it take to bind variants) was removed as this information is rarely provided in studies. This resulted in a stable set of eight dimensions that we used for the remainder of the pilot. During data collection for this pilot, we excluded primary studies that do not focus on particular aspects of variability and their use in software systems, such as taxonomies. We also added a few additional options to some of the dimensions. For example, we added "scenario" as an option for artifact and "code generation" as an option for realization technique.

5. The initial set of dimensions included requirement type, variability introduction phase, anticipation of variability, time of binding, frequency of binding, trigger for variability, artifact affected by variability, variability realization technique, binding entity, time to bind, locality of variability, predictability of variability, impact of variability, criticality of variability, openness, orthogonality, and uniformity of variability.

TABLE 9
Dimensions of Variability

Type **Introduction and specification of variability**	
Dimension	**Domain (number of studies in pilot)**
Requirement type	Functional (15) Quality (17)
Representation	Feature model (4) Rules/conditions (15) Variant labels/annotations (7) Profiles (4) Change scenarios (2)
Artifact	Scenario (5) Business process (1) Architecture (14) Component (14) Code fragment (5) Variable (2)
Orthogonality	Separated (15) Integrated (7)
Mechanisms **How does variability take place**	
Trigger	Stakeholder (14) Business process (1) System (5) Environment (8)
Realization technique	Reorganization (7) Selection (18) Value assignment (2) Code generation (2)
Time of binding	Software construction/evolution (10) Runtime (13)
Automation	Manual (3) Semi-automatic (10) Automatic (9)

4.3.2 Overview of the Dimensions

Table 9 gives an overview of the dimensions. For each dimension, we provide different options ("domain") and the number of studies of the pilot assigned to an option. A detailed overview of the dimensions with examples from the pilot is provided in Tables 10 and 11.

The eight dimensions are grouped in two clusters. The first cluster, *type of variability*, refers to the introduction and specification of variability. Based on the data collected in the pilot, we observe that the requirements that variability addresses are approximately equally divided between functional and quality requirements (requirement type). The dominating approaches to represent variability are rules/conditions and variant labels/annotations in software artifacts (representation). The primary elements that are subject of variability are architecture and component (artifact), and

the majority of the studies specify variability in separated distinct artifacts (orthogonality).

The second cluster, *mechanisms of variability*, refers to the way variability is actually realized. The trigger to bind variants mostly comes from stakeholders, followed by the environment of the system (trigger). The dominant technical approaches for the realization of variability are selection of variants followed by reorganization of system artifacts (realization technique). The time when variants are bound is approximately equally distributed over software construction/evolution and runtime (time of binding), and almost all studies provide some kind of tool support to bind variants to variation points (automation).

4.3.3 Evaluation of Dimensions

Evaluating a classification framework is in general a difficult task. A systematic evaluation, such as the one proposed by Gomez-Perez [57], would require a formal specification of our classification framework, which is beyond the scope of this study. Based on [57], we discuss three important evaluation criteria of the proposed classification: consistency, completeness, and expandability. Consistency refers to whether it is possible to obtain contradictory conclusions from applying the dimensions. After an initial phase of the pilot, we defined accurate descriptions of the dimensions and their options were straightforward to apply. While there are obvious relationships between the dimensions, based on our experiences with collecting the data during the pilot, we can conclude that the dimensions cover disjoint but complementary facets of variability. Completeness refers to the coverage of the dimensions, and dimensions that may have been overlooked. As explained, the intended scope of the classification is not to be complete, but to define a set of core dimensions to describe and compare

TABLE 10
Dimensions of Variability—Type Cluster

Requirement type – the type of requirement that variability should address
Functional: Variability addresses a function of a software system (or a part of it), i.e., a required functionality in terms of inputs, behavior, and outputs.
Study 11 supports different types of missions of a command and controller simulator that are provided by optional components. In study 42, a workflow of a wine production process is extended with new variants of particular regions in the process to accommodate the production of new wines differently branded.
Quality: Variability addresses a quality property of a system (or a part of it), that is, a non-functional property that indicates how well the system delivers its functionality.
Study 47 supports interoperability of a web service implementation by a reconfigurable proxy that dynamically selects, tests and deploys a proper set of adaptors when a new service is bound to the service API. In study 2, an alternative service is selected and invoked when the original service is not available.
Representation – the way variability is described
Feature model: Variability is represented as a compact, typically graphical description of all the valid products of a family of systems in terms of combinations of features.
Study 118 uses a feature graph and links between features and requirements specifications to model requirements variability of a product line.
Rules/conditions: Variability is represented by (or the representation is supported by) a set of rules or conditions that refer to elements or artifacts that document/realize the system or family of systems.
Study 13 uses logical expressions and conditions that refer to elements of design patterns to express variants of the patterns. Study 146 employs decisions – which are defined by a name, type, condition, attributes, and effects – to describe variation points in a product line and choices to derive products.
Variant labels/annotations: Variability is represented by (or the representation is supported by) labels or annotations added to artifacts that document/realize the system or family of systems.
Study 54 annotates components with properties (e.g., required resources) to discriminate between alternatives, enabling dynamic selection of proper variants with highest utility for a given context.
Profiles: Variability is represented by profiles that provide descriptive summaries of users or artifacts in the environment (e.g., in the form of a table, model or a set of expressions). Profiles are typically combined with other types of representations.
Study 81 employs meta-data (composition policy) together with a description of contextual information to dynamically select and add services (extensions) to application components on a per usage basis (e.g., add authentication).
Change scenarios: Variability is represented by (or the representation is supported by) change scenarios that describe events, sequences of events or options and how they affect a change of a system or family of systems.
In study 92, architects map a set of strategic scenarios (i.e., possible developments of the world relevant to the domain) to a set of architecture scenarios, each specifying a specific set of choices in the variation model of a view; analysis of the scenarios supports decision making for variability of the product line.

TABLE 10
Continued

Artifact – element subject to variability
Scenario: The subject of variability is a scenario related to any aspect of the system or family of systems.
Study 24 distinguishes scenario variants based on annotated information, supporting evolution and traceability of scenarios. Study 28 employs a variability meta model to document requirements artifacts across product lines; the requirements artifacts specify the abstract variants by using scenarios among others.
Business process: The subject of variability is a business process, i.e., a collection of related, structured activities or tasks that produce a specific service or product.
Study 42 dynamically adapts/updates a workflow by selecting and replacing a new variant of a region in the workflow.
Architecture: The subject of variability is the software architecture of the system or family of systems.
Study 17 compares two or more candidate software architectures and selects the optimal candidate for different change scenarios. In study 38, a chain of services is dynamically reconfigured to the most optimal configuration according to an adaptation policy.
Component: The subject of variability is a component of the software system or family of systems.
In study 37, a service can be dynamically adapted by selecting a replacement from a pool of services that fits best the current context. Study 59 discusses an imaging server that offers images to clients, where a variant of the requested image is selected (scaled) with optimal resolution based on the quality of client connections, arrival rate of requests, and CPU load.
Code fragment: The subject of variability is a fragment of source code of the system or family of systems.
Study 73 dynamically updates applications by adding, removing, or replacing methods of classes based on the difference between different variants of the classes.
Variable: The subject of variability is a variable defined in the system or family of systems.
Study 54 supports parameterization of program variables allowing fine-grained adaptation at runtime. Study 146 supports fine-grained variability in a steel plant automation system (among other types) with configuration variables for features such as colors and speed.
Orthogonality – separation of variability specification from the software systems
Separated: Variability is specified in distinct artifacts, separated from the artifacts that document/realize the system or family of systems.
Study 146 describes variability with a decision model (comprising a set of decisions, see rules/conditions option of representation) together with an asset model that describes the solution space to derive products in terms of assets, such as components, resources, etc.
Integrated: The specification of variability is interwoven within other artifacts that document/realize a system or family of systems.
Study 51 associates variant labels to data elements of a database to capture the variants to which the elements apply. In study 11, variability is embedded in the implementation of the classes by means of an inheritance refinement hierarchy and state machines.

key facets of variability. From our experiences with the review in general and the pilot in particular, we conclude that the classification covers a representative set of core dimensions for describing variability. As we used only a subset of primary studies, the ranges of some of the dimensions will not be complete. For example, more options for realization technique may appear when the dimensions will be applied to other studies. This brings us to expandability, which refers to the ability for extending the classification (i.e., adding new dimensions to the classification and new options to existing dimensions), without altering the set of well-defined properties. The "open" nature of the proposed classification allows researchers to easily add new dimensions or options for dimensions.

4.3.4 Usefulness of the Classification

The classification offers a vocabulary for researchers and engineers that allows specifying and comparing the variability properties under consideration in their studies and projects. The dimensions can be used both during system realization and reverse engineering activities. The dimensions support software engineers with handling and describing the key facets of variability in their systems explicitly, which is crucial to make informed decisions.

The classification also enables comparison of variability solutions across different systems and different domains. If extended to a full classification that covers all primary studies in this literature review, the classification would be a great asset for identifying related work in a specific area of variability, and identifying specific areas in variability research that deserve further attention. We leave this effort for future work.

In conclusion, we believe that this empirically grounded classification provides a solid step towards a unifying, integrated perspective of variability in software systems, spanning across currently disparate or loosely coupled research themes in the software engineering community. Such unifying perspective has the potential to accelerate the exchanges of ideas across research themes, reducing duplicate efforts, and encouraging best practices.

As an example, we notice a high potential for exchanging ideas between researchers from the areas of software product lines and dynamic service adaptation. Applications of variability in dynamic service adaptation offer solutions that can be exploited to postpone variant binding in product lines until runtime. Likewise, dynamic service adaptation can exploit models for representing variability from the area of software product lines. Two other areas where the exchange of ideas offers interesting opportunities for research are variability in business processes and variability in software architecture.

4.4 Variability Handling in Software Engineering versus Variability Management in SPL

Some of our findings confirm results from recent reviews on variability management in software product line engineering [19], [20], [58]. For example, no study on variability-related issues has been published in empirical venues. On the other hand, reviews on variability management in product lines found that the premium event of product line engineering (SPLC) does not have a clear dominance when it comes to studies on variability. In contrast, we found that SPLC is the dominating conference for publishing variability-related research. This difference could be due to the fact that Chen [19] performed an automated search on many sources. This caused the results being spread across a large number of venues. Furthermore, Chen and Babar's study has a smaller scope, i.e., variability management instead of handling variability.

Also, similar to the reviews on variability management in product lines, we found that most current work does not provide sufficient support for the claimed utility of proposed approaches and uses less rigorous evaluation. This could also be a reason why no studies on variability have been published in empirical venues.

Finally, approaches for variability management in product lines share significant commonalities (e.g., the notion of feature and feature modeling). However, we did not find such commonalities in the context of variability handling in software engineering. Thus, a reference model for variability handling as suggested by Chen and Babar might be difficult to achieve in software engineering in general.

TABLE 11
Dimensions of Variability—Mechanism Cluster

Trigger – the source that initiates the binding of variants to variation points
Stakeholder: Variant binding is initiated by a stakeholder, i.e., a person with an interest in the system or family of systems.
In study 56, user commands trigger the selection of variants of GUI forms that guide a user through a process of generating legal documents.
Business process: A business process triggers variant binding.
In study 42, the firing conditions of a transition in a business process trigger the execution of a variant of a region of the process.
System: Variant binding is triggered by the system.
In study 5, a service system optimizes its workflow by replacing services with services of better quality when the current quality of service differs from the prediction by more than a given threshold.
Environment: An artifact in the environment triggers variant binding.
In study 54, changes of the context, including computing resources, quality of the network, and physical location trigger the system to replace components by variants that guarantee better utility.
Realization technique – technical approach to realize variability
Reorganization: Variability is realized by reorganizing the structure or behavior of the system artifacts.
In study 38, environmental changes trigger a configuration manager to dynamically reconfigure a service chain according to an adaptation policy. In study 42, different variants of a region in a business process can be triggered for execution, each determining a different flow in the execution of the business process.
Selection: Variability is realized by selection of a variant among a set of variants and binding that variant to a variation point.
In study 118, product line requirements are specified using annotations that represent features. Feature selection results in product-specific requirements specifications. In study 5, when a service invocation fails, a new service is selected among available variants that optimizes the overall quality of service of the composite service.
Value assignment: Variability is realized by assigning a value to a variation point.
In study 146, values are assigned to configuration parameters of a steel plant automation system for features such as colors, UI appearance, speed, and amount.
Code generation: Variability is realized by generating the code of a variant that is bound to a variation point.
Study 51 employs variant labels attached to data elements of a database or relations between data elements to generate variant specific code for a product of an avionic product line.

TABLE 11
Continued

Time of binding – time when variants are bound to variation points
Software construction/evolution: Variants are bound to variation points during system construction (design, realization, instantiation), or evolution.
In study 13, a designer selects alternatives and options specified in a design pattern to derive a pattern variant for the problem at hand. In study 17, an analyst selects the best architecture among a set of variants during system development based on the impact analysis for a set of modifiability scenarios.
Runtime: Variants are bound to variation points during system operation.
Study 55 supports dynamic content adaptations of mobile devices based on device profiles and users' service level agreements by selecting matching services from a repository of variants.
Automation – degree of support to automate variant binding
Manual: Variant binding is a manual effort.
In study 92, architects select a specific set of choices in the variation models of different architectural views for different change scenarios; these choices provide variants that are used for strategic decision making about supported variability.
Semi-automatic: Variant binding is performed manually but supported by tools.
Study 24 provides automation support for managing scenario evolution by enabling users to define new scenarios and select a scenario among the available variants. In study 28, a tool supports users to model requirements variability across a set of product lines and select variability information for a product line of interest.
Automatic: Variant binding is fully automated.
In study 81, interceptors intercept component interactions and dynamically determine which services (extensions) need to be selected and deployed based on a composition policy and context information of the current execution.

5 Deviations from Review Guidelines, Threats to Validity and Assessment of Review

5.1 Deviations from Systematic Literature Review Guidelines

Even though we followed Kitchenham's and Charters' procedures for systematic reviews [5], we deviated from these guidelines in two ways.

First, the search was performed as a manual search, including a set of specific journals and conference proceedings, rather than using an automated search. We believe that this is a valid deviation: As argued by Brereton et al. [10], current search engines for software engineering venues are not designed to support systematic literature reviews. Using a manual search is consistent with strategies followed by other researchers (such as [11]). Compared to [11], our study includes almost twice as many venues as well as covers a broader time range. A more detailed justification for the manual search was presented in Section 2. However, a limitation of the targeted manual search is that we may have missed some relevant studies published at venues that we did not include in our search, and therefore provided an incomplete insight into variability in software systems. In particular, we have missed articles published in national journals or conferences, or workshops. We also have missed articles in conferences aimed at specific software engineering topics, outside the generic phases of software engineering. We have also omitted an extensive search of technical reports and theses as good quality grey literature would eventually appear as journal or conference papers. Thus, our results must be qualified as applying to studies published in major international software engineering journals and conferences. To mitigate the problem of missing studies published in major international software engineering journals and conferences, we conducted a limited targeted automated search.

Second, for each study a single researcher extracted the data and another researcher checked the data extraction. Even though this deviates from Kitchenham's and Charters' guidelines, this practice has been suggested by Brereton et al. [10]. A consequence of our approach is that some of the data that we collected may be erroneous. As outlined by Turner et al. [59], such an extractor/checker mode can lead to problems in cases where a large number of studies has to be analyzed. This is particularly true for the quality assessment, as the evaluation of quality criteria may be subjective. On the other hand, having several extractors per paper is even less feasible for a large number of studies, and having no checker per paper would have caused more erroneous data.

5.2 Threats to Validity

First, the review process assumed a common understanding among all reviewers about the search and analysis methods. Misunderstandings of concepts could potentially cause biased results. This threat was mitigated by having the review protocol provided to all reviewers and discussed before the start of the review to ensure a selection process as unbiased as possible.

Second, during the review we found that some papers insufficiently describe approaches or did not provide

enough information to appropriately collect the data as outlined in the protocol. Therefore, we had to infer certain pieces of information during the data extraction process. To minimize the possibility of introducing inaccuracy in the extracted data, we a) recorded data as presented in the study, and b) discussed among researchers to clarify ambiguity during the review process.

Third, we limited the search by starting in the year 2000. This may affect the completeness of our search results as our review does not include research published before the year 2000. However, as shown in a previous study, only a very small number of papers on variability management has been published in the product line domain before the year 2000 [20]. Furthermore, we limited the search based on selected venues using the ranking of the Australian Research Council and the H-index. Different ranking and quality indicators could lead to different venues included in the search.

Fourth, we could have collected additional information, e.g., about the scalability of approaches, or the size of models used in approaches. However, this kind of data is often not reported in studies and thus would require speculation.

5.3 Assessment of Review

We evaluate our systematic review against a set of four quality questions for systematic reviews proposed by Kitchenham et al. [11]:

1. Are inclusion and exclusion criteria described and appropriate? This criterion is met as we explicitly defined and explained inclusion and exclusion criteria in the study.
2. Is the literature search likely to have covered all relevant studies? According to Kitchenham et al., this criterion would have been met if either four or more digital libraries and additional search strategies had been identified, or if all journals addressing the topic of interest had been identified. As we cannot claim that we included all relevant journals, but specified a restricted set of journals and conferences, we consider this criterion as partially met. We could, for example, have included conference venues from the agile software engineering domain.
3. Did the reviewers assess the quality / validity of the included studies? We consider this criterion as met as we have explicitly defined quality criteria and carefully selected search venues based on their quality. We extracted quality criteria from each primary study, instead of merely formulating research questions which target quality issues.
4. Were the basic data / studies adequately described? We consider that this criterion is met as we used a detailed data collection form for each study. This data collection form was piloted.

6 Conclusions

We performed a systematic literature review to study variability handling in software engineering to support variability in software systems. Our focus was on research studies reported in primary software engineering journals and conferences.

The study results show that many methods for variability handling still lack evidence for the validity of proposed approaches. Thus, to increase evidence and to make studies more attractive for practitioners, we recommend the following:

1. Researchers should provide more data to support findings and to convince practitioners about the validity of new proposals.
2. To allow other researchers to understand approaches properly, details on the research design should be provided, including elaborations on the credibility of findings.
3. More empirical experiments and industrial studies should be performed, not only to increase validity of proposals, but also to address needs of industrial software development.

We employed the results of the literature review to provide a unifying, integrated perspective of variability in software systems. The resulting classification of variability in different dimensions as outlined in Section 4.3 captures key facets of variability, allowing researchers and engineers to identify and compare their approaches for variability handling in different types of software systems. The empirically grounded set of core dimensions offers a basis that researchers and engineers can extend with additional dimensions and new options towards defining a comprehensive classification of key properties of variability in software systems.

Based on the study findings, we point to the following interesting opportunities for future research:

1. Research on handling variability with respect to quality attributes (beyond performance and availability) is required.
2. Further research is required to develop disciplined approaches for handling variability in open service-based systems.
3. Testing and verification in the context of variability handling are interesting areas for future research.
4. A classification of the complete set of primary studies of this literature review using the dimensions introduced in Section 4.3 would provide a solid basis for identifying related work in sub-fields of variability handling.
5. To further enhance our understanding of variability handling, this review could be complemented by looking at other research areas that employ principles from variability modeling, such as literature on knowledge modeling in the artificial intelligence community and feature interaction that can also be seen as a way to manage or handle variability.

Finally, we believe that the strength of our review lies in a) the manual search of selected quality venues and journals, complemented by a targeted automated search to increase reliability of results and reproducibility, and b) the broad perspective on variability which we could get through the manual search. However, we do not share the experience reported by Kitchenham et al. that manual search takes less effort than an automated search [35].

APPENDIX

TABLE 12
Sources of Searched Venues

Venue	Source
ASEJ	SpringerLink
ESE	SpringerLink
IEEE SW	IEEE Xplore
IST	ScienceDirect
JSS	ScienceDirect
REJ	SpringerLink
SCP	ScienceDirect
SMRP	Wiley Online Library
SoSyM	SpringerLink
SPE	Wiley Online Library
STVR	Wiley Online Library
TOSEM	ACM Digital Library
TSE	IEEE Xplore
ASE	ACM Digital Library, IEEE Xplore
CAiSE	SpringerLink, DBLP
EASE	IEEE Xplore (special issue of IET Software), BCS
ECOOP	SpringerLink
ECSA	SpringerLink, IEEE Xplore
FSE	ACM Digital Library
GPCE	DBLP, ACM Digital Library, SpringerLink
ICSE	ACM Digital Library, IEEE Xplore
ICSM	IEEE Xplore
ICST	IEEE Xplore
ISSTA	ACM Digital Library
MODELS	SpringerLink, DBLP
OOPSLA	ACM Digital Library
QoSA	SpringerLink
RE	IEEE Xplore
SLE	SpringerLink, DBLP
SPLC	ACM Digital Library, DBLP, IEEE Xplore, SpringerLink
WICSA	IEEE Xplore

TABLE 13
Number of Journal Papers Searched, and Papers Left After Each Search Stage

Venue	Papers searched	Papers after Stage 1	Papers after Stage 2
ASEJ	168	13	5
ESE	306	7	1
IEEE SW	1509	53	14
IST	1098	39	24
JSS	1729	78	38
REJ	217	15	6
SCP	647	34	19
SMRP	232	28	15
SoSyM	263	13	9
SPE	759	41	17
STVR	128	1	0
TOSEM	171	7	4
TSE	764	40	16
Sum	**7991**	**368**	**168**

TABLE 14
Number of Conference Papers Searched, and Papers Left After Each Search Stage

Venue	Papers searched	Papers after Stage 1	Papers after Stage 2
ASE	713	39	19
CAiSE	489	59	13
EASE	116	7	2
ECOOP	337	3	1
ECSA	168	25	10
FSE	452	20	10
GPCE	221	45	5
ICSE	1206	43	21
ICSM	887	22	11
ICST	214	3	3
ISSTA	233	2	0
MODELS	542	46	8
OOPSLA	422	4	2
QoSA	91	10	5
RE	639	30	11
SLE	87	12	4
SPLC	325	97	79
WICSA	297	37	18
Sum	**7439**	**504**	**222**

TABLE 15
Studies Included in the Final Review

ID	Authors	Year	Title	Venue
1	L. Andrade et al.	2001	Enforcing business policies through automated reconfiguration	ASE
2	V. Antonellis et al.	2006	A layered architecture for flexible web service invocation	SPE
3	L. Apvrille et al.	2004	Verifying service continuity in a dynamic reconfiguration procedure: application to a satellite system	ASEJ
4	D. Ardagna et al.	2007	PAWS: a framework for executing adaptive web-service processes	IEEE SW
5	D. Ardagna, B. Pernici	2007	Adaptive service composition in flexible processes	TSE
6	M. Ardis et al.	2000	Software product lines: a case study	SPE
7	C. Atkinson et al.	2000	Component-based product line development: the KobrA approach	SPLC
8	M. Ali Babar et al.	2010	Managing variability in software product lines	IEEE SW
9	L. Baresi et al.	2006	Style-based modeling and refinement of service-oriented architectures	SoSyM
10	I. Barone et al.	2008	COMOVER: Concurrent model versioning	ICSM
11	D. Batory et al.	2002	Achieving extensibility through product-lines and domain-specific languages: a case study	TOSEM
12	D. Batory et al.	2002	Generating product-lines of product-families	ASE
13	I. Bayley, H. Zhu	2010	Formal specification of the variants and behavioural features of design patterns	JSS
14	P. Bellavista et al.	2003	Context-aware middleware for resource management in the wireless Internet	TSE
15	S. Mokhtar et al.	2007	COCOA: COnversation-based service COmposition in pervAsive computing environments with QoS support	JSS
16	H. Benestad et al.	2009	Understanding software maintenance and evolution by analyzing individual changes: a literature review	SMRP

TABLE 15
Continued

17	P. Bengtsson et al.	2004	Architecture-level modifiability analysis (ALMA)	JSS
18	I. Ben-Shaul et al.	2001	Dynamic adaptation and deployment of distributed components in Hadas	TSE
19	D. Beuche et al.	2007	Using requirements management tools in software product line engineering: the state of the practice	SPLC
20	D. Beuche et al.	2004	Variability management with feature models	SCP
21	P. Boinot et al.	2000	A declarative approach for designing and developing adaptive components	ASE
22	J. Bosch	2004	On the development of software product-family components	SPLC
23	A. Braganca, R. Machado	2006	Extending UML 2.0 metamodel for complementary usages of the <<extend>> relationship within use case variability specification	SPLC
24	K. Breitman et al.	2005	Supporting scenario evolution	REJ
25	P. Brito et al.	2009	Verifying architectural variabilities in software fault tolerance techniques	ECSA
26	A. Brogi et al.	2006	On the semantics of software adaptation	SCP
27	A. Brogi et al.	2006	Component adaptation through flexible subservicing	SCP
28	S. Buhne et al.	2005	Modelling requirements variability across product lines	RE
29	G. Canfora et al.	2008	A framework for QoS-aware binding and re-binding of composite web services	JSS
30	J. Cao et al.	2003	Dynamic configuration management in a graph-oriented distributed programming environment	SCP
31	J. Cao et al.	2006	An interactive service customization model	IST
32	M. Caporuscio et al.	2007	Model-based system reconfiguration for dynamic performance management	JSS
33	L. Capra et al.	2003	CARISMA: context-aware reflective middleware system for mobile applications	TSE
34	V. Cardellini et al.	2009	Qos-driven runtime adaptation of service oriented architectures	FSE
35	H. Cervantes, R. Hall	2004	Autonomous adaptation to dynamic availability using a service-oriented component model	ICSE
36	C. Cetina et al.	2009	Strategies for variability transformation at run-time	SPLC
37	T. Chaari et al.	2007	A comprehensive approach to model and use context for adapting applications in pervasive environments	JSS
38	A. Chan, C. Siu-Nam	2003	MobiPADS: a reflective middleware for context-aware mobile computing	TSE
39	S. Chang, S. Kim	2007	A variability modeling method for adaptable services in service-oriented computing	SPLC
40	L. Chen et al.	2009	Variability management in software product lines: a systematic review	SPLC
41	M. Chu-Carroll et al.	2002	Supporting aggregation in fine grained software configuration management	FSE
42	F. Cicirelli et al.	2010	A service-based architecture for dynamically reconfigurable workflows	JSS
43	M. Coriat et al.	2000	The SPLIT method: building product lines for software-intensive systems	SPLC
44	C. Costa et al.	2007	Dynamic reconfiguration of software architectures through aspects	ECSA
45	C. Courbis, A. Finkelstein	2005	Towards aspect weaving applications	ICSE
46	S. Deelstra et al.	2009	Variability assessment in software product families	IST
47	G. Denaro et al.	2009	Ensuring interoperable service-oriented systems through engineered self-healing	FSE
48	D. Dhungana et al.	2008	Supporting the evolution of product line architectures with variability model fragments	WICSA
49	G. Modica et al.	2009	Dynamic SLAs management in service oriented environments	JSS
50	E. Nitto et al.	2008	A journey to highly dynamic, self-adaptive service-based applications	ASEJ
51	F. Dordowsky, W. Hipp	2009	Adopting software product line principles to manage software variants in a complex avionics system	SPLC
52	N. Duzbayev, I. Poernomo	2007	Pre-emptive adaptation through classical control theory	QoSA
53	Y. Eracar, M. Kokar	2000	An architecture for software that adapts to changes in requirements	JSS
54	J. Floch et al.	2006	Using architecture models for runtime adaptability	IEEE SW
55	M. Forte et al.	2008	Using ontologies and web services for content adaptation in ubiquitous computing	JSS
56	C. Fritsch, B. Renz	2004	Four mechanisms for adaptable systems	SPLC
57	K. Fung, G. Low	2009	Methodology evaluation framework for dynamic evolution in composition-based distributed applications	JSS
58	C. Ghezzi et al.	2010	QoS driven dynamic binding in-the-many	QoSA
59	I. Gorton et al.	2008	An extensible and lightweight architecture for adaptive server applications	SPE
60	A. Gruler et al.	2007	Development and configuration of service-based product lines	SPLC
61	J. van Gurp et al.	2010	Comparing practices for reuse in integration-oriented software product lines and large open source software projects	SPE
62	S. Hallsteinsen et al.	2006	Using product line techniques to build adaptive systems	SPLC
63	A. Harhurin, J. Hartmann	2008	Service-oriented commonality analysis across existing systems	SPLC
64	R. Hirschfeld, K. Kawamura	2006	Dynamic service adaptation	SPE
65	A. van der Hoek et al.	2001	Taming architectural evolution	FSE
66	M. Karam et al.	2008	A product-line architecture for web service-based visual composition of web applications	JSS
67	M. Koning et al.	2009	VxBPEL: supporting variability for web services in BPEL	IST
68	J. Lee, K. Kang	2006	A feature-oriented approach to developing dynamically reconfigurable products in product line engineering	SPLC

TABLE 15
Continued

69	J. Lee, G. Kotonya	2010	Combining service-orientation with product line engineering	IEEE SW
70	J. Lee et al.	2008	An approach for developing service oriented product lines	SPLC
71	E. Niemelä, A. Immonen	2007	Capturing quality requirements of product family architecture	IST
72	C. Parra et al.	2009	Context awareness for dynamic service-oriented product lines	SPLC
73	S. Previtali, T. Gross	2006	Dynamic updating of software systems based on aspects	ICSM
74	R. Schantz et al.	2006	Controlling quality-of-service in distributed real-time and embedded systems via adaptive middleware	SPE
75	M. Sinnema, S. Deelstra	2007	Classifying variability modeling techniques	IST
76	M. Sinnema et al.	2004	COVAMOF: a framework for modeling variability in software product families	SPLC
77	C. Sun et al.	2010	Modeling and managing the variability of web service-based systems	JSS
78	H. Sun et al.	2009	Product-line-based requirements customization for web service compositions	SPLC
79	M. Svahnberg et al.	2005	A taxonomy of variability realization techniques	SPE
80	S. Thiel, A. Hein	2002	Systematic integration of variability into product line architecture design	SPLC
81	E. Truyen et al.	2001	Dynamic and selective combination of extensions in component-based applications	ICSE
82	J. van Gurp, J. Savolainen	2006	Service grid variability realization	SPLC
83	Y. Wang, J. Mylopoulos	2009	Self-repair through reconfiguration: a requirements engineering approach	ASE
84	I. Warren et al.	2006	An automated formal approach to managing dynamic reconfiguration	ASE
85	E. Wohlstadter et al.	2004	GlueQoS: middleware to sweeten quality-of-service policy interactions	ICSE
86	S. Yau et al.	2008	Specification, decomposition and agent synthesis for situation-aware service-based systems	JSS
87	H. Zhang, S. Jarzabek	2004	XVCL: a mechanism for handling variants in software product lines	SCP
88	A. Zisman et al.	2008	A framework for dynamic service discovery	ASE
89	N. Aguirre, T. Maibaum	2002	A temporal logic approach to the specification of reconfigurable component-based systems	ASE
90	S. Ajila, Ali Kaba	2008	Evolution support mechanisms for software product line process	JSS
91	V. Alagar et al.	2003	A rigorous approach for constructing self-evolving real-time reactive systems	IST
92	P. America et al.	2004	Scenario-based decision making for architectural variability in product families	SPLC
93	T. Asikainen et al.	2006	A unified conceptual foundation for feature modelling	SPLC
94	P. Bachara et al.	2010	Framework for application management with dynamic aspects J-EARS case study	IST
95	L. Baresi et al.	2004	Style-based refinement of dynamic software architectures	WICSA
96	M. Bernardo et al.	2002	Architecting families of software systems with process algebras	TOSEM
97	J. Bosch	2002	Maturity and evolution in software product lines: approaches, artefacts and organization	SPLC
98	G. Botterweck et al.	2008	Visual tool support for configuring and understanding software product lines	SPLC
99	A. Braganca, R. Machado	2007	Automating mappings between use case diagrams and feature models for software product lines	SPLC
100	T. Brown et al.	2002	Adaptable components for software product line engineering	SPLC
101	A. Bucchiarone et al.	2009	Self-repairing systems modeling and verification using AGG	WICSA
102	P. Buhr, W. Mok	2000	Advanced exception handling mechanisms	TSE
103	R. Capilla, M. Ali Babar	2008	On the role of architectural design decisions in software product line engineering	ECSA
104	C. Cetina et al.	2008	Applying software product lines to build autonomic pervasive systems	SPLC
105	F. Chauvel et al.	2010	Using QoS-contracts to drive architecture-centric self-adaptation	QoSA
106	H. Chu et al.	2004	Roam, a seamless application framework	JSS
107	L. Chung, N. Subramanian	2003	Architecture-based semantic evolution of embedded remotely controlled systems	SMRP
108	J. Cobleigh et al.	2002	Containment units: a hierarchically composable architecture for adaptive systems	FSE
109	A. Colman, J. Han	2007	Using role-based coordination to achieve software adaptability	SCP
110	C. Costa-Soria et al.	2008	Managing dynamic evolution of architectural types	ECSA
111	E. Curry, P. Grace	2008	Flexible self-management using the model-view-controller pattern	IEEE SW
112	K. Czarnecki et al.	2008	Sample spaces and feature models: there and back again	SPLC
113	S. Deelstra et al.	2005	Product derivation in software product families: a case study	JSS
114	J. Dehlinger, R. Lutz	2006	PLFaultCAT: a product-line software fault tree analysis tool	ASEJ
115	J. Dehlinger, R. Lutz	2008	Supporting requirements reuse in multi-agent system product line design and evolution	ICSM
116	C. Rosso	2008	Software performance tuning of software product family architectures: two case studies in the real-time embedded systems domain	JSS
117	D. Dhungana et al.	2010	Structuring the modeling space and supporting evolution in software product line engineering	JSS
118	M. Eriksson et al.	2009	Managing requirements specifications for product lines - an approach and industry case study	JSS
119	J. Estublier, G. Vega	2005	Reuse and variability in large software applications	FSE
120	L. Etxeberria, G. Sagardui	2008	Variability driven quality evaluation in software product lines	SPLC
121	A. Fantechi, S. Gnesi	2008	Formal modeling for product families engineering	SPLC
122	S. Faulk	2001	Product-line requirements specification (PRS): an approach and case study	RE

TABLE 15
Continued

123	Q. Feng, R. Lutz	2005	Bi-directional safety analysis of product lines	JSS
124	E. Figueiredo et al.	2008	Evolving software product lines with aspects	ICSE
125	A. Fortier et al.	2010	Dealing with variability in context-aware mobile software	JSS
126	G. Gannod, R. Lutz	2000	An approach to architectural analysis of product lines	ICSE
127	A. Garg et al.	2003	An environment for managing evolving product line architectures	ICSM
128	K. Geihs et al.	2009	A comprehensive solution for application-level adaptation	SPE
129	M. Goedicke et al.	2004	Designing runtime variation points in product line architectures: three cases	SCP
130	H. Gomaa, M. Hussein	2004	Software reconfiguration patterns for dynamic evolution of software architectures	WICSA
131	B. Gonzales-Baixauli, et al.	2004	Visual variability analysis for goal models	RE
132	A. Gregersen, B. Jørgensen	2009	Dynamic update of Java applications - balancing change flexibility vs programming transparency	SMRP
133	R. Gumzej et al.	2009	A reconfiguration pattern for distributed embedded systems	JSS
134	A. Hallerbach et al.	2010	Capturing variability in business process models: the Provop approach	SMRP
135	G. Halmans, K. Pohl	2003	Communicating the variability of a software-product family to customers	JSS
136	M. Abu-Matar, H. Gomaa	2011	Variability modeling for service oriented product line architectures	SPLC
137	R. Ali et al.	2010	A goal-based framework for contextual requirements modeling and analysis	REJ
138	A. C. Contieri et al.	2011	Extending UML components to develop software product-line architectures: lessons learned	ECSA
139	P. Asirelli et al.	2011	Formal description of variability in product families	SPLC
140	R. Baird et al.	2011	Self-adapting workflow reconfiguration	JSS
141	J. Ferreira Bastos et al.	2011	Adopting software product lines: a systematic mapping study	EASE
142	T. Berger et al.	2010	Variability modeling in the real: a perspective from the operating systems domain	ASE
143	R. Cavalcanti et al.	2011	Extending the RiPLE-DE process with quality attribute variability realization	QoSA
144	L. Chen, M. Ali Babar	2011	A systematic review of evaluation of variability management approaches in software product lines	IST
145	K. Czarnecki, M. Antkiewicz	2005	Mapping features to models: a template approach based on superimposed variants	GPCE
146	D. Dhungana et al.	2011	The DOPLER meta-tool for decision-oriented variability modeling: a multiple case study	ASEJ
147	M. Erwig	2010	A language for software variation research	GPCE
148	M. Erwig, E. Walkingshaw	2011	The choice calculus: a representation for software variation	TOSEM
149	N. Esfahani, S. Malek	2010	On the role of architectural styles in improving the adaptation support of middleware platforms	ECSA
150	J. Feigenspan et al.	2011	Using background colors to support program comprehension in software product lines	EASE
151	M. Galster, P. Avgeriou	2011	Handling variability in software architecture: problems and implications	WICSA
152	N. Gui et al.	2011	Toward architecture-based context-aware deployment and adaptation	JSS
153	A. Haber et al.	2011	Hierarchical variability modeling for software architectures	SPLC
154	A. Heuer et al.	2010	Formal definition of syntax and semantics for documenting variability in activity diagrams	SPLC
155	C. Kaestner et al.	2011	Variability-aware parsing in the presence of lexical macros and conditional compilation	OOPSLA
156	S. Kato, N. Yamaguchi	2011	Variation management for software product lines with cumulative coverage of feature interactions	SPLC
157	J. Liebig, et al.	2010	An analysis of the variability in forty preprocessor-based software product lines	ICSE
158	X. Peng et al.	2011	Analyzing evolution of variability in a software product line: from contexts and requirements to features	IST
159	M. Rosenmueller et al.	2011	Tailoring dynamic software product lines	GPCE
160	M. Rosenmueller et al.	2011	Flexible feature binding in software product lines	ASEJ
161	U. Ryssel	2010	Automatic variation-point identification in function-block-based models	GPCE
162	K. Schmid	2010	Variability modeling for distributed development – a comparison with established practice	SPLC
163	J. Sincero et al.	2010	Efficient extraction and analysis of preprocessor-based variability	GPCE
164	C. Thörn	2010	Current state and potential of variability management practices in software-intensive SMEs: Results from a regional industrial survey	IST
165	M. Vierhauser et al.	2010	Flexible and scalable consistency checking on product line variability models	ASE
166	D. Weyns et al.	2011	An architectural approach to support online updates of software product lines	WICSA
167	M. Acher et al.	2010	Composing feature models	SLE
168	M. Alferez et al.	2010	Multi-view composition language for software product line requirements	SLE
169	V. Andrikopoulos et al.	2008	Managing the evolution of service specifications	CAiSE
170	D. Benavides et al.	2005	Automated reasoning on feature models	CAiSE
171	J. Bergh, K. Coninx	2006	CUP 2.0: High-level modeling of context-sensitive interactive applications	MODELS
172	K. Boukadi et al.	2008	An aspect oriented approach for context-aware service domain adapted to e-business	CAiSE

TABLE 15
Continued

173	F. Casati et al.	2000	Adaptive and dynamic service composition in eFlow	CAiSE
174	M. Cengarle et al.	2009	Variability within modeling language definitions	MODELS
175	F. Dalpiaz et al.	2009	An architecture for requirements-driven self-reconfiguration	CAiSE
176	C. Dorn, S. Dustdar	2010	Interaction-driven self-adaptation of service ensembles	CAiSE
177	F. Fleurey, A. Solberg	2009	A domain specific modeling language supporting specification, simulation and execution of dynamic adaptive systems	MODELS
178	G. Halmans et al.	2008	Documenting application-specific adaptations in software product line engineering	CAiSE
179	P. Jayaraman et al.	2007	Model composition in product lines and feature interaction detection using critical pair analysis	MODELS
180	P. Lahire et al.	2007	Introducing variability into aspect-oriented modeling approaches	MODELS
181	B. Morin et al.	2009	Weaving variability into domain metamodels	MODELS
182	A. Reuys et al.	2005	Model-based system testing of software product families	CAiSE
183	S. Shiraishi	2010	An AADL-based approach to variability modeling of automotive control systems	MODELS
184	A. Shui et al.	2005	Exceptional use cases	MODELS
185	B. Weber et al.	2007	Change patterns and change support features in process-aware information systems	CAiSE
186	S. Zschaler et al.	2010	VML*– a family of languages for variability management in software product lines	SLE
187	K. Bak et al.	2011	Feature and meta-models in clafer: mixed, specialized, and coupled	SLE
188	S. Liaskos et al.	2011	Goal-based behavioral customization of information systems	CAiSE
189	B. Pernici, S. Siadat	2011	A fuzzy service adaptation based on QoS satisfaction	CAiSE
190	N. Qureshi et al.	2011	Requirements engineering for self-adaptive systems: core ontology and problem statement	CAiSE
191	M. Weidlich et al.	2011	A foundational approach for managing process variability	CAiSE
192	M. Mezini, K. Ostermann	2004	Variability management with feature-oriented programming and aspects	FSE
193	D. Webber, H. Gomaa	2004	Modeling variability in software product lines with the variation point model	SCP
194	A. Classen et al.	2010	Model checking lots of systems : efficient verification of temporal properties in software product lines	ICSE
195	A. Classen et al.	2011	A text-based approach to feature modelling : syntax and semantics of TVL	SCP
196	S. Kim et al.	2005	A theoretical foundation of variability in component-based development	IST

ACKNOWLEDGMENTS

The authors are grateful to the anonymous reviewers and the TSE editor for the thorough feedback on earlier versions of this paper. Their comments and suggestions were of great help. This work was partially supported by the Marie Curie CIG number 303791 and by NWO SaS-LeG, contract no. 638.000.000.07N07.

REFERENCES

[1] J. van Gurp, J. Bosch, and M. Svahnberg, "On the Notion of Variability in Software Product Lines," *Proc. Working IEEE/IFIP Conf. Software Architecture*, pp. 45-54, 2001.

[2] L. Chen and M.A. Babar, "Variability Management in Software Product Lines: An Investigation of Contemporary Industrial Challenges," *Proc. 14th Int'l Software Product Line Conf.*, pp. 1-15, 2010.

[3] R. Hilliard, "On Representing Variation," *Proc. Workshop Variability in Software Product Line Architectures*, pp. 312-315, 2010.

[4] M. Galster and P. Avgeriou, "Handling Variability in Software Architecture: Problems and Implications," *Proc. Ninth IEEE/IFIP Working Conf. Software Architecture*, pp. 171-180, 2011.

[5] B. Kitchenham and S. Charters, "Guidelines for Performing Systematic Literature Reviews in Software Enginnering," technical report, Keele Univ., Keele, United Kingdom, 2007.

[6] M. Staples and M. Niazi, "Experiences Using Systematic Review Guidelines," *J. Systems and Software*, vol. 80, no. 9, pp. 1425-1437, Sept. 2007.

[7] J. Biolchini et al., "Systematic Review in Software Engineering,"Technical Report RT - ES 679/05, Programa de Engenharia de Sistemas e Computacao, Rio de Janeiro, Brazil, 2005.

[8] M. Riaz et al., "Experiences Conducting Systematic Reviews from Novices' Perspective," *Proc. 14th Int'l Conf. Evaluation and Assessment in Software Eng. (EASE '10)*, pp. 1-10, 2010.

[9] H. Zhang and M.A. Babar, "On Searching Relevant Studies in Software Engineering," *Proc. 14th Int'l Conf. Evaluation and Assessment in Software Eng. (EASE '10)*, pp. 1-10, 2010.

[10] P. Brereton et al., "Lessons from Applying the Systematic Literature Review Process within the Software Engineering Domain," *J. Systems and Software*, vol. 80, no. 4, pp. 571-583, Apr. 2007.

[11] B. Kitchenham et al., "Systematic Literature Reviews in Software Engineering—A Systematic Literature Review," *Information and Software Technology*, vol. 51, no. 1, pp. 7-15, 2009.

[12] B. Kitchenham et al., "Systematic Literature Reviews in Software Engineering—A Tertiary Study," *Information and Software Technology*, vol. 52, no. 8, pp. 792-805, Aug. 2010.

[13] B. Williams and J. Carver, "Characterizing Software Architecture Changes: A Systematic Review," *Information and Software Technology*, vol. 52, no. 1, pp. 31-51, Jan. 2010.

[14] K. Schmid and I. John, "A Customizable Approach to Full Lifecycle Variability Management," *Science of Computer Programming*, vol. 53, no. 3, pp. 259-284, Dec. 2004.

[15] M. Aiello, P. Bulanov, and H. Groefsema, "Requirements and Tools for Variability Management," *Proc. Fourth IEEE Workshop Requirements Eng. for Services (REFSQ '10)*, pp. 245-250, 2010.

[16] L. Etxeberria and G. Sagardui, "Evaluation of Quality Attribute Variability in Software Product Families," *Proc. 15th Ann. IEEE Int'l Conf. Eng. Computer Based Systems (ECBS)*, pp. 255-264, 2008.

[17] J. van Gurp and J. Bosch, "Preface," *Proc. Software Variability Management Workshop*, p. 1, 2003.

[18] M. Svahnberg, J. van Gurp, and J. Bosch, "A Taxonomy of Variability Realization Techniques," *Software—Practice and Experience*, vol. 35, no. 8, pp. 705-754, Apr. 2005.

[19] L. Chen, M.A. Babar, and N. Ali, "Variability Management in Software Product Lines: A Systematic Review," *Proc. 13th Int'l Software Product Line Conf. (SPLC)*, pp. 81-90, 2009.

[20] L. Chen and M.A. Babar, "A Systematic Review of Evaluation of Variability Management Approaches in Software Product Lines," *Information and Software Technology*, vol. 53, no. 4, pp. 344-362, 2011.

[21] A. Kontogogos and P. Avgeriou, "Towards Modelling Variability-Intensive SOA Systems,"technical report, Univ. of Groningen, The Netherlands, 2009.

[22] F. Bachmann et al., "A Meta-Model for Representing Variability in Product Family Development," *Proc. Fifth Int'l Workshop Software Product Family Eng.*, pp. 66-80, 2003.
[23] K. Pohl, G. Boeckle, and F. van der Linden, *Software Product Line Engineering—Foundations, Principles, and Techniques*. Springer Verlag, 2005.
[24] R. Kazhamiakin et al., *Adaptation of Service-Based Systems*, M.P Papazoglou et al., eds., pp. 117-156, Springer Verlag, 2010.
[25] V. Alves et al., "Requirements Engineering for Software Product Lines: A Systematic Literature Review," *Information and Software Technology*, vol. 52, no. 8, pp. 806-820, Aug. 2010.
[26] R. Rabiser, P. Gruenbacher, and D. Dhungana, "Requirements for Product Derivation Support: Results from a Systematic Literature Review and an Expert Survey," *Information and Software Technology*, vol. 52, no. 3, pp. 324-346, Mar. 2010.
[27] D. Benavides, S. Segura, and A. Ruiz-Cortes, "Automated Analysis of Feature Models 20 Years Later: A Literature Review," *Information Systems*, vol. 35, no. 6, pp. 615-636, 2010.
[28] A. Hubaux et al., "A Preliminary Review on the Application of Feature Diagrams in Practice," *Proc. Fourth Int'l Workshop Variability Modelling of Software-Intensive Systems*, pp. 53-59, 2010.
[29] M.S. Ali et al., "A Systematic Review of Comparative Evidence of Aspect-Oriented Programming," *Information and Software Technology*, vol. 52, no. 9, pp. 871-887, 2010.
[30] B. Kitchenham, "Procedures for Performing Systematic Reviews," technical report, Keele Univ., Keele, United Kingdom, 2004.
[31] V. Basili, G. Caldiera, and D. Rombach, "The Goal Question Metric Approach," *Encyclopedia of Software Eng.*, J.J. Marciniak, ed., pp. 528-532, John Wiley & Sons, 1994.
[32] L. Etxeberria and G. Sagardui, "Variability Driven Quality Evaluation in Software Product Lines," *Proc. 12th Int'l Software Product Line Conf.*, pp. 243-252, 2008.
[33] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Addison-Wesley, 2003.
[34] B. Kitchenham et al., "The Impact of Limited Search Procedures for Systematic Literature Reviews: A Participant-Observer Case Study," *Proc. Third Int'l Symp. Empirical Software Eng. and Measurement*, pp. 336-345, 2009.
[35] B. Kitchenham et al., "Refining the Systematic Literature Review Process—Two Participant-Observer Case Studies," *Empirical Software Eng.*, vol. 15, no. 6, pp. 618-653, 2010.
[36] M. Jorgensen and M. Shepperd, "A Systematic Review of Software Development Cost Estimation Studies," *IEEE Trans. Software Eng.*, vol. 33, no. 1, pp. 33-53, Jan. 2007.
[37] L. Chen, M.A. Babar, and C. Cawley, "A Status Report on the Evaluation of Variability Management Approaches," *Proc. 13th Int'l Conf. Evaluation and Assessment in Software Eng. (EASE)*, pp. 1-10, 2009.
[38] Australian Research Council, "Ranked Outlets," http://www.arc.gov.au/era/era_2010/era_2010.htm, Aug. 2010.
[39] M. Galster, et al., "Variability in Software Systems—Extracted Data and Supplementary Material from a Systematic Literature Review," Technical Report, Univ. of Groningen, Groningen, The Netherlands, 2013.
[40] B. Kitchenham et al., "An Evaluation of Quality Checklist Proposals—A Participant-Observer Case study," *Proc. 13th Int'l Conf. Evaluation and Assessment in Software Eng. (EASE)*, pp. 1-10, 2009.
[41] T. Dyba and T. Dingsoyr, "Empirical Studies of Agile Software Development: A Systematic Review," *Information and Software Technology*, vol. 50, nos. 9/10, pp. 833-859, Aug. 2008.
[42] T. Dyba, T. Dingsoyr, and G.K. Hanssen, "Applying Systematic Reviews to Diverse Study Types: An Experience Report," *Proc. Int'l Symp. Empirial Software Eng. and Measurement*, pp. 225-234, 2007.
[43] B. Cheng et al., "Software Engineering for Self-Adaptive Systems: A Research Roadmap," *Software Eng. for Self-Adaptive Systems*, B. Cheng, et al, ed., et al., pp. 1-26, Springer Verlag, 2009.
[44] M. Papazoglou et al., "Service-Oriented Computing: State of the Art and Research Challenges," *Computer*, vol. 40, no. 11, pp. 38-45, Nov. 2007.
[45] S. Hallsteinsen et al., "Dynamic Software Product Lines," *Computer*, vol. 41, no. 4, pp. 93-95, Apr. 2008.
[46] *IEEE Standard Glossary of Software Engineering Terminology*, Standard IEEE Std 610.12-1990, 1990.
[47] *Software Engineering - Product Quality—Part 1: Quality Model*, Standard ISO/IEC 9126-1, 2001.
[48] A. Gehlert and A. Metzger, *Quality Reference Model for SBA*, S-Cube, 2009.
[49] C. Zannier, G. Melnik, and F. Maurer, "On the Success of Empirical Studies in the International Conference on Software Engineering," *Proc. 28th Int'l Conf. Software Eng.*, pp. 341-350, 2006.
[50] D. Weyns et al., "Claims and Supporting Evidence for Self-Adaptive Systems: A Literature Study, Software Engineering for Adaptive and Self-Managing Systems," *Proc. Int'l Symp. Software Eng. for Adaptive and Self-Managing Systems (SEAMS)*, pp. 89-98, 2012.
[51] C. Sun et al., "Modeling and Managing the Variability of Web-Service-Based Systems," *J. Systems and Software*, vol. 83, no. 3, pp. 502-516, Mar. 2010.
[52] E. Engstroem and P. Runeson, "Software Product Line Testing—A Systematic Mapping Study," *Information and Software Technology*, vol. 53, no. 1, pp. 2-13, 2011.
[53] P. Neto et al., "A Systematic Mapping Study of Software Product Line Testing," *Information and Software Technology*, vol. 53, no. 5, pp. 407-423, 2011.
[54] M. Ivarsson and T. Gorschek, "A Method for Evaluating Rigor and Industrial Relevance of Technology Evaluations," *Empirical Software Eng.*, vol. 16, no. 3, pp. 365-395, 2011.
[55] S. Mahdavi-Hezavehi, M. Galster, and P. Avgeriou, "Variability in Quality Attributes of Service-Based Software Systems: A Systematic Literature Review," *Information and Software Technology*, vol. 55, no. 2, pp. 320-343, 2013.
[56] J. Andersson et al., "Modeling Dimensions of Self-Adaptive Software Systems," *Software Eng. for Self-Adaptive Systems*, B. Cheng et al., eds., pp. 27-47, Springer Verlag, 2009.
[57] A. Gomez-Perez, "Evaluation of Ontologies," *Int'l J. Intelligent Systems*, vol. 16, no. 3, pp. 391-409, 2001.
[58] M.A. Babar, L. Chen, and F. Shull, "Managing Variability in Software Product Lines," *IEEE Software*, vol. 27, no. 3, pp. 89-91, May/June 2010.
[59] M. Turner et al., "Lessons Learnt Undertaking a Large-Scale Systematic Literature Review," *Proc. 12th Int'l Conf. Evaluation and Assessment in Software Eng. (EASE '08)*, pp. 1-10, 2008.
[60] M. Galster and P. Avgeriou, "Variability in Web Services," in *Systems and Software Variability Management*, R. Capilla, J. Bosch, and K. Kang, eds., pp. 269-278, Springer Verlag, 2013.

Matthias Galster is a lecturer at the University of Canterbury, Christchurch, New Zealand.

Danny Weyns is a professor at Linnaeus University, Sweden.

Dan Tofan is a doctoral student at the University of Groningen, The Netherlands. He also works as a software engineer in Belgium.

Bartosz Michalik received the PhD degree from the Katholieke Universiteit Leuven, Belgium. He is currently a senior software developer at Amartus, Poland.

Paris Avgeriou is a professor and the head of the Software Engineering and Architecture group at the University of Groningen, The Netherlands.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.