University of Groningen

Assumptions and their management in software development

Yang, Chen; Liang, Peng; Avgeriou, Paris

Published in:
Information and Software Technology

DOI:
[10.1016/j.infsof.2017.10.003](https://doi.org/10.1016/j.infsof.2017.10.003)

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
2018

[Link to publication in University of Groningen/UMCG research database](#)

Citation for published version (APA):

Yang, C., Liang, P., & Avgeriou, P. (2018). Assumptions and their management in software development: A systematic mapping study. *Information and Software Technology*, 94, 82-110.
<https://doi.org/10.1016/j.infsof.2017.10.003>

Copyright

Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

The publication may also be distributed here under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license. More information can be found on the University of Groningen website: <https://www.rug.nl/library/open-access/self-archiving-pure/taverne-amendment>.

Take-down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): <http://www.rug.nl/research/portal>. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.

Assumptions and their management in software development: A systematic mapping study

Chen Yang^{a,b}, Peng Liang^{a,*}, Paris Avgeriou^b

^a State Key Laboratory of Software Engineering, School of Computer Science, Wuhan University, 430072 Wuhan, China

^b Department of Mathematics and Computing Science, University of Groningen, Nijenborgh 9, 9747 AG Groningen, The Netherlands

ARTICLE INFO

Keywords:

Software development
Assumption
Assumptions management
Systematic mapping study

ABSTRACT

Context: Assumptions are constantly made by stakeholders or generated automatically in software development. However, there is a lack of systematic analysis and comprehensive understanding of the research and practice regarding assumptions and their management.

Objective: This work aims to explore and analyze the state of the art on assumptions and their management in software development.

Method: A systematic mapping study that covers the literature from January 2001 to December 2015 on assumptions and their management in software development.

Results: 134 studies were included: (1) The studies were published in 94 venues, which indicates that assumptions and their management has been a broad topic in software engineering. (2) Only 21 studies defined the assumption concept. (3) Most assumptions are made for or related to the artifacts in requirements engineering and software design, which demonstrates that assumptions should be managed from the early phases of software development. (4) Much effort has been put on Assumptions Making, Description, and Evaluation. Assumptions Maintenance received moderate attention. More than half of the tools identified aim to support assume-guarantee reasoning. For the other tools, most of them can be used to support Assumptions Description. (5) All the identified types of stakeholders are involved in Assumptions Making, followed by Evaluation and Description. Stakeholders involved in requirements engineering, software design, and software construction play a central role in assumptions management. (6) The main challenge is the difficulty of performing assumptions management activities in software development. (7) The identified assumptions management approaches, tools, benefits, and lessons learned are limited to their specific contexts (e.g., context of use). (8) Most of the negative consequences are caused by invalid or implicit assumptions.

Conclusions: This work provides researchers and practitioners with a reflection of the past fifteen years of research and practice on assumptions and their management in software development.

1. Introduction

An assumption is defined as “a thing that is accepted as true or as certain to happen, without proof”¹ or as “a fact or statement taken for granted”.² Accordingly, we define software assumption as software development knowledge taken for granted or accepted as true without evidence. This definition of assumption emphasizes the characteristic of uncertainty in software development knowledge: stakeholders believe but cannot know for sure the importance, impact, or correctness of specific software development knowledge (e.g., requirements knowledge, such as priority of requirements [12] and design knowledge, such

as design decisions [12]). Moreover, as defined by Kroll and Kruchten [45]: “An artifact is a piece of information that is produced, modified, or used by a process”. According to this definition, assumptions are a type of software artifact, similarly to requirements, design decisions, etc.

Assumptions in the field of software development is a broad topic: different types of assumptions (e.g., requirement assumptions [9], software design assumptions [8], and software construction assumptions [10]) have been extensively discussed. For instance, in the early phases of software development, there could be many uncertain things. However, in order to meet the project business goals (e.g., schedule and deadlines), stakeholders have to work in the presence of such

* Corresponding author.

E-mail address: liangp@whu.edu.cn (P. Liang).

¹ <http://www.oxforddictionaries.com/definition/english/assumption>.

² <http://www.merriam-webster.com/dictionary/assumption>.

uncertainties; these uncertainties can lead to assumptions. In this paper, we advocate treating uncertainty and assumption as two different but related concepts: one way to deal with uncertainties is to make implicit or explicit assumptions (i.e., a thing that is uncertain, but accepted as true).

Many studies are related to uncertainty in software development, but to the best of our knowledge, there are no secondary studies on this topic. Therefore, we present four related studies as representative examples. Ziv et al. mentioned that there are various types of uncertainty in software development, such as uncertainty in requirements analysis, transition from system requirements to design and code, software re-engineering, software reuse, and testing [40]. For example, they referred to Humphrey's book [41] to explain why requirements analysis has uncertainties: *"For a new software system, the requirements will not be completely known until after the users have used it"*. Liu et al. [42] focused on requirements uncertainty, and discussed the relationships among requirements uncertainty, interpersonal conflict, and software project performance. For example, they explained that requirements uncertainties lead to unstable requirements, and such requirements can further result in interpersonal conflicts between stakeholders. Whittle et al. [43] proposed a language (RELAX) to address requirements uncertainty in the context of self-adaptive systems. They considered uncertainties from two sources: (1) environmental uncertainty (i.e., changing environmental conditions), and (2) behavioral uncertainty (i.e., changing requirements). Cheng et al. [44] also mentioned that environmental uncertainty is a major motivation for dynamic adaptation (i.e., continuously adapting behaviors of a system according to its changing environment). They stated that: *"It is impossible to know at development time all of the possible combinations of environmental conditions that will be encountered"*.

Many researchers have pointed out the importance of assumptions and their management in software development. For example, Corbató [18] mentioned in his ACM Turing Award lecture that *"design bugs are often subtle and occur by evolution with early assumptions being forgotten as new features or uses are added to systems."* Another example is from our recent survey on software architectural assumptions with 112 practitioners working in IT industry [11]: architectural assumptions are critical in both software development and architecture design. Various problems in software development can be traced to not well-managed assumptions [14]; we provide four examples of such problems. Garlan et al. pointed out that incompatible assumptions in software architecture could lead to architectural mismatch [5]. Moreover, stakeholders may misunderstand assumptions, because the assumption concept is rather subjective [24]; this can further lead to misunderstandings of other software artifacts that are related to such assumptions [24]. Steingruebl and Peterson [15] argued that undocumented software assumptions could lead to software failures, and they proposed methods and techniques to manage software assumptions. Finally, Bazaz et al. [16] defined vulnerability as *"a state of the system from which it is possible to transition to an incorrect system state"*; they pointed out that the violation of assumptions about system resources might cause the system to be vulnerable. More details regarding these problems and their consequences are provided in Section 2.

Given the importance of assumptions and their management in software development, there is a need for systematic analysis and comprehensive understanding of the research and practice regarding this topic. To the best of our knowledge, there is no such systematic review. Mamun and Hansson [13] conducted a review of assumptions in software development, but the review was based on their knowledge without following a systematic approach (e.g., to search and select the related studies) [4]. To fill this gap, we conducted a systematic mapping study to explore available evidence as well as spot gaps regarding nine aspects of assumptions and their management (see Section 3.1 for more details). This work provides researchers and practitioners with a reflection of the past fifteen years of research and practice on this topic.

The methods of systematic literature review (SLR) and systematic

mapping study (SMS) are typically used to conduct secondary studies [4]. An SLR aims at identifying, evaluating, and interpreting all available research regarding a certain research question, topic area, or phenomenon of interest [4]. An SMS focuses on providing a wide overview of a domain, identifying research evidence on a topic, and presenting mainly quantitative results [4]. One of the main differences between an SLR and SMS is that SLRs focus on precise research questions, while SMSs have a broader scope [4]. Assumptions and their management in software development is a broad topic, which covers many different aspects (e.g., software development activities, artifacts, and stakeholders). Thus, as suggested by Kitchenham and Charters [4], we decided to conduct an SMS instead of an SLR.

The rest of the paper is structured as follows. Section 2 introduces the context of the SMS, while Section 3 describes the SMS design. The answers to each research question (RQ) are provided in Section 4. Sections 5 and 6 discuss the results and the threats to the validity respectively. Section 7 concludes this work.

2. Context

The core concepts of this SMS are assumptions and their management in software development. This section first introduces assumptions regarding their characteristics and classifications as well as the relations between assumptions and other types of software artifacts. Then it elaborates on assumptions management based on several aspects.

2.1. Characteristics of assumptions

As mentioned in Section 1, every assumption has uncertainty. Uncertainty is an important criterion to judge whether a piece of information is an assumption. In our previous studies regarding architectural assumptions and their management (e.g., [11,24]), we found that the assumption concept is subjective (e.g., whether a piece of information is an assumption), which is a major reason that stakeholders may have different understandings of the assumption concept. For example, Lago and van Vliet [25] mentioned that it is difficult to draw the line between architectural assumptions, requirements, and constraints. Furthermore, though stakeholders can understand the assumption concept, they rarely use the *"assumption"* term in their work [23,24]. Instead, they mix assumptions with other types of software artifacts, such as requirements, design decisions, and risks [23,24]. Assumptions have a dynamic nature, i.e., they can evolve over time [14,23]. For example, during software development, a valid assumption can turn out to be invalid or vice versa, or an assumption can transform to another type of software artifact or vice versa. Furthermore, assumptions are context dependent [23,24]. For example, the same assumption could be valid in one project, while invalid in another project because the context changes; or an assumption in one project is not an assumption in another project. The dynamic and context-dependent nature could be one reason that not-well managed assumptions lead to a multitude of problems in software development. As an example, the reuse of an assumption from ARIANE 4, which was not valid in ARIANE 5, led to the ARIANE 5 disaster [30].

2.2. Classifications of assumptions

There are various classifications of assumptions in software development. For example, Garlan et al. [5] classified assumptions that can lead to architectural mismatch into four types: the nature of the components, the nature of the connectors, the global architectural structure, and the construction process (development environment and build). Lago and van Vliet [22] classified architectural assumptions as technical, organizational, and management assumptions. However, most of the existing classifications are related to a limited set of assumptions (e.g., the classification proposed by Garlan et al. is only for architectural

assumptions). To the best of our knowledge, there is no general classification of assumptions in software development (e.g., based on software development activities).

2.3. Assumptions and other types of software artifacts

Assumptions are not independent in software development, but intertwined with many types of software artifacts. For example, when managing assumptions in software design (e.g., [26,27]), assumptions are usually related to requirements, design decisions, components, etc. However, as mentioned in Point (2) of this section, the existing studies only focus on limited types of assumptions (e.g., requirement assumptions [9]). Therefore, there is a lack of knowledge regarding which software artifacts are related to assumptions in general, as well as how they are related.

2.4. Assumptions management

Assumptions management is an ill-defined concept. Though there are certain approaches and tools (e.g., [8,28]) for managing assumptions, there is no accepted definition of assumptions management and no overview of the supporting approaches and tools. Furthermore, assumptions management should be teamwork, i.e., different types of stakeholders should be involved [23,29]. However, there is a lack of evidence regarding who should be involved, as well as how they are involved in assumptions management. Though assumptions were found important in software development, the investment and return of managing assumptions needs more evidence (e.g., which benefits and challenges stakeholders will face when managing assumptions in software development).

Assumptions can be managed both manually and automatically. In the latter case, formal approaches and tools are used to automatically manage assumptions (e.g., assumptions are usually made and documented by such approaches and tools instead of stakeholders). A representative example of such management is assume-guarantee reasoning in system verification (e.g., [31,32,33,34]). On the other side, manual assumptions management refers to manual work by stakeholders. For example, stakeholders use approaches and tools to make and document assumptions (e.g., [8,9,35,36]). More discussion between automatic and manual assumptions management can be found in Section 5.2.

3. Mapping study design

This section introduces the objective, RQs, and execution of the SMS.

3.1. Objective and research questions

The objective of this SMS was formulated based on the Goal-Question-Metric approach [3]: **analyze primary studies for the purpose of exploration and analysis with respect to assumptions and their management from the point of view of researchers and practitioners in the context of software development.**

To get a detailed and comprehensive view of the study topic, the objective of this SMS was decomposed into nine RQs, as shown in Table 1: an understanding of assumptions in software development (RQ1, RQ2, and RQ3), assumptions management activities, approaches, and tools (RQ4 and RQ5), the stakeholders, benefits, challenges, and lessons learned of assumptions management (RQ6, RQ7, and RQ9), and the consequences caused by not well-managed assumptions (RQ8). Each RQ is mapped to the discussion in Section 2.

3.2. Mapping study execution

This SMS followed the guidelines proposed by Petersen et al. [20]. This section discusses the process employed in the mapping study,

including the trial search and selection process (see Section 3.2.1), formal search and selection process (see Section 3.2.2), data extraction (see Section 3.2.3), and data analysis (see Section 3.2.4).

3.2.1. Trial search and selection

During the trial search and selection process we encountered a number of problems. These problems as well as how we handled them, are elaborated one by one:

(1) How to search in different databases?

The trial search and selection helped the authors to find appropriate search methods for each database (see Table 2) because databases employed various search engines with different search capabilities. For example, some databases provided options to narrow research areas to Computer Science and languages to English; this improved the efficiency of the subsequent steps.

(2) Which search terms to use?

The trial search and selection were used to refine the search terms in the search query. We followed a number of steps to identify an appropriate query expression:

- Several synonyms of “assumption” can be identified in an English dictionary. However, these synonyms do not always have the same semantics with the term “assumption”, as considered in the scope of our work (see Section 2). For example, many papers use “guess” or “hypothesis”, but unless the papers explicitly mention that these terms are equal to “assume” or “assumption”, we cannot conclude that they have the same meaning.
- To ground the search in the context of software development, the query expression “(software engineering OR software development OR system development) AND (...)” was initially used. However, we found that many qualified studies only mention, for example, “software” in their titles or abstracts, instead of “software engineering” or “software development”.
- We further tried the following query expression “(assume OR assuming OR assumption)”, “system AND (assume OR assuming OR assumption)”, “program AND (assume OR assuming OR assumption)”, and “(requirement OR design OR architecture OR component OR source code OR testing) AND (assume OR assuming OR assumption)”. However, the total numbers of retrieved papers were rather enormous. For example, by searching in the IEEE Explore database (one of the seven databases used in the SMS) through the query expression “(assume OR assuming OR assumption)” from 2001 to 2015, the number of retrieved papers was 82,650.
- To balance the value of the SMS and the effort needed, we chose to use the query expression “(software) AND (assumption OR assume OR assuming)” in the SMS. We note that this could lead to a threat of missing relevant studies, and this threat is discussed in Section 6.

(3) How to decide for inclusion/exclusion of studies in the first round of selection?

In a number of occasions, only reading the title of a paper could not help to make a decision regarding whether the paper should be included or not. Therefore, we decided to read the title and abstract of a paper simultaneously in the formal search and selection (i.e., Phase 2 in Fig. 2).

(4) Which papers to include?

The trial search and selection were used to refine the selection criteria for the formal search and selection. For example, two types of papers were identified in the trial search and selection, i.e., papers regarding assumptions in software development and assumptions about an approach or a tool. Moreover, we found some papers that either mention the term “*assumption*” without actually using it as meant in this study or contain no relevant data to answer the RQs. To avoid losing focus, we decided to only include the papers that concern assumptions in software development, and exclude papers that do not have enough data related to the RQs (see Section 3.2.2.4).

Table 1
Research questions of this mapping study, their rationale, and related discussion.

Research questions	Rationale	Discussion
RQ1: What are the definitions of assumption in software development?	Researchers and practitioners may have different understandings of assumptions in software development. The aim of this RQ is to collect such data to show how assumptions are treated and defined in software development.	Point (1) in Section 2
RQ2: What are the types of assumptions in software development?	Different types of assumptions have been discussed in various software development activities (e.g., software design). The answer to this RQ provides an overview of various types of assumptions in software development.	Point (2) in Section 2
RQ3: Which software artifacts are related to assumptions in software development?	Assumptions are not independent in software development, but interact with various types of software artifacts. The aim of this RQ is to know which artifacts are related to assumptions.	Points (1) and (3) in Section 2
RQ4: Which activities have been proposed to support assumptions management in software development?	Assumptions management contains a set of activities. The aim of this RQ is to identify the assumptions management activities that have been employed or discussed in the literature.	Point (4) in Section 2
RQ5: Which approaches and tools are available to support assumptions management in software development?	There are different approaches and tools that can support assumptions management in software development. The answer to this RQ can make researchers and practitioners be aware of these specific approaches and tools.	Point (4) in Section 2
RQ6: Which stakeholders are involved in assumptions management in software development?	Assumptions management is related to various stakeholders (e.g., architects). The results of this RQ can help researchers and practitioners to understand who need to be involved and how they play a role in assumptions management.	Point (4) in Section 2
RQ7: What are the benefits and challenges of assumptions management in software development?	Assumptions management in software development leads to certain benefits with challenges. The answer to this RQ can make researchers and practitioners be aware of such benefits and challenges when employing assumptions management in software development.	Point (4) in Section 2
RQ8: What are the consequences when assumptions are not well managed in software development?	Not well-managed assumptions (e.g., implicit assumptions) may cause problems. The results of this RQ make researchers and practitioners be aware of the negative impact caused by not well-managed assumptions in software development.	Points (1) to (4) in Section 2
RQ9: What are the lessons learned from assumptions management in software development?	Lessons learned refer to the experience from the authors of the studies about managing assumptions in software development. The answer of this RQ can help researchers and practitioners to get such experience.	Points (1) to (4) in Section 2

Table 2
Databases used in the search.

Database	Link	Matched scope in database
ACM Digital Library	http://dl.acm.org/	Title, abstract
IEEE Explore	http://ieeexplore.ieee.org/Xplore/home.jsp	Title, keywords, abstract
Science Direct	http://www.sciencedirect.com/	Title, keywords, abstract
Springer Link	http://link.springer.com/	Title
Wiley InterScience	http://onlinelibrary.wiley.com/	Title, abstract
EI Compendex	https://www.engineeringvillage.com/	Title, abstract
ISI Web of Science	https://login.webofknowledge.com	Title, keywords, abstract

(5) How to achieve a joint understanding among authors?

The trial search and selection helped the authors to reach a consensus on various aspects of the study search and selection. For example, to reach an agreement on the selection criteria, a sample of 100 papers were chosen for further discussion among the authors, regarding why they should or should not be included.

The procedure of the trial search and selection is shown in [Fig. 1](#). One researcher (R1) searched and selected papers in IEEE Explore, while another researcher (R2) searched and selected papers in Wiley InterScience and ISI Web of Science. The results of the search and selection were reviewed by a third researcher.

3.2.2. Formal search and selection

This section introduces the procedure of the formal search and selection (see [Section 3.2.2.1](#)), search scope (see [Section 3.2.2.2](#)), search query (see [Section 3.2.2.3](#)), and selection criteria (see [Section 3.2.2.4](#)).

3.2.2.1. Procedure

The execution procedure of the formal search and selection in seven phases is shown in [Fig. 2](#).

Phase 1: Searching papers in seven databases (see [Table 2](#)).

Phase 2: Conducting the 1st round selection (i.e., by title and abstract) based on the results of the database search.

Phase 3: Selecting papers from the results of Phase 2 by reading the full text.

Phase 4: Using the Snowballing technique [\[6\]](#) to manually check all the references of the selected papers from Phase 3. Snowballing entails using the reference list of a paper (backward Snowballing) or the citations to the paper (forward Snowballing) in order to identify additional studies [\[6\]](#). We conducted backward Snowballing, collecting all the references from the papers resulting from Phase 3 (i.e., Phases 4–1), then selected papers first by title and abstract (i.e., Phases 4–2), and subsequently by full text (i.e., Phases 4–3).

Phase 5: Searching and selecting papers in the seven databases regarding rely-guarantee approaches and assumption-commitment approaches. The reason that we conducted the extended search and selection is that the results of Phases 3 and 4 showed that some qualified studies manage assumptions in software development using assume-guarantee approaches, which are also called rely-guarantee or assumption-commitment approaches.

Phase 6: Extracting data (see [Table 3](#)) of the selected papers (including a trial data extraction).

Phase 7: Analyzing the extracted data from Phase 6.

3.2.2.2. Search scope

Based on our experience in conducting secondary studies [\[7\]](#), and according to other systematic reviews on the topics of software engineering [\[1,19\]](#), the databases listed in [Table 2](#) were used as the sources for the database search of this SMS. Since the seven databases used different search engines and strategies, the matched scopes were different. For example, in the Springer Link database,³ the search engine allows users to search “with all of the words”, “with the exact phrase”, “with at least one of the words”, “without the words”, “where the title contains”, “where the authors / editor is”, and “show documents published”.

³ <https://link.springer.com/advanced-search>.

Fig. 1. Process of the trial search and selection.

It does not support searching in keywords or abstract. Therefore, the matched scope in the Springer Link database was only “Title”.

The time period of search was set between January 2001 and December 2015 (i.e., the starting time of this SMS). Searching without a temporal limitation would be extremely resource-intensive, and could not be achieved in a realistic time frame. However, to the best of our knowledge, there are no papers that can potentially act as a milestone in this domain (i.e., assumptions and their management in software development), because the topic is general and broad. Therefore, we set the start year to 2001, considering that 15 years is a reasonable period for the topic of this SMS. Moreover, Google Scholar was not included in the database search, because the precision of the retrieved results from Google Scholar is insufficient, and it may have considerable overlap with other databases such as IEEE Explore on software engineering literature [2].

3.2.2.3. Search query

Boolean “OR” was used to join alternative words and synonyms and Boolean “AND” was used to join major terms:

- (1) Phase 1: “(software) **AND** (assumption **OR** assume **OR** assuming)”
- (2) Phase 5: “(rely guarantee **OR** rely/guarantee **OR** rely-guarantee)”
- (3) Phase 5: “(assumption commitment **OR** assumption-commitment **OR** assumption/commitment)”

3.2.2.4. Selection criteria

Inclusion criterion:

I1: The paper concerns assumptions in software development (e.g., requirement assumptions).

Exclusion criteria:

E1: The paper concerns assumptions about an approach or a tool.

E2: The paper is gray literature (e.g., technical report) [4].

E3: If the same work was published in more than one venue, the less mature papers are excluded.

E4: The paper is not written in English.

E5: The paper only has an abstract and not a full text.

E6: The paper merely mentions the term “assumption”.

E7: The paper contains no relevant data to answer the RQs.

E1 was used because, to the best of our knowledge, there exist two types of assumptions in software development: (1) assumptions made in software development (e.g., assuming that the end users of the software are mostly elder people) and (2) assumptions made for a specific approach or tool (e.g., making assumptions about the design of an approach or tool). Since the topic of this SMS is assumptions and their management in software development, E1 was set to exclude all the papers related to the second type of assumptions.

3.2.3. Data extraction

Table 3 shows the details (i.e., description and related RQ) of the extracted data items in this SMS.

3.2.4. Data analysis

The extracted data were analyzed based on the RQs. We used a bubble chart (see Fig. 7) to visualize types of assumptions (see Section 4.3), assumptions management activities (see Section 4.5), and time period. For answering RQ1, RQ2, RQ3, RQ4, and RQ5, both descriptive statistics and Constant Comparison (i.e., generating concepts from the extracted data) [17] were employed. For answering the other

Fig. 2. Process of the formal search and selection.

RQs, we only employed Constant Comparison.

Constant Comparison provides a systematic way to generate concepts from data, and a continuous process of verification of the generated concepts and categories [17,21]. We followed the guidelines proposed by Adolph et al. [21] to conduct Constant Comparison. We coded the extracted data as incidents, compared these incidents to each other to generate concepts, and further performed comparison among incidents and concepts to generate categories. For example, if a paper mentions: “Sometimes changes could be adopted easily; but sometimes even crucial parts of the application had to be rewritten, often because its

assumptions did not hold any longer”, the incident was first coded as “Consequences caused by not well-managed assumptions”. We also used subcodes to detail the codes of incidents. Considering the aforementioned example, the part of the incident mentioning “often because its assumptions did not hold any longer”, was sub-coded as “Consequences caused by not well-managed assumptions: Invalid assumptions”. Note that the length of an incident can vary from a single word to several paragraphs. Furthermore, the Constant Comparison process was iteratively conducted. Finally, we did not predefine codes, but let the codes gradually emerge during the Constant Comparison process.

Table 3
Data items to be extracted.

#	Data item	Description	RQ
D1	ID	The ID of the study	N/A
D2	Title	The title of the study	N/A
D3	Author	The authors of the study	N/A
D4	Type of authors	The type of authors (i.e., academia, industry, or both)	N/A
D5	Publication type	The type of publication of the study (e.g., journal)	N/A
D6	Publication venue	The name of the venue where the study was published	N/A
D7	Publication year	The publication year of the study	N/A
D8	Definition	The definitions of assumption	RQ1
D9	Software development activity	The software development activities (e.g., software design) that involve assumptions	RQ2
D10	Artifact	The artifacts (e.g., requirements) related to assumptions in software development	RQ3
D11	Assumptions management activity	The assumptions management activities (e.g., Assumptions Making)	RQ4
D12	Approach	The assumptions management approaches	RQ5
D13	Tool	The assumptions management tools	RQ5
D14	Stakeholder	The stakeholders who are involved in assumptions management	RQ6
D15	Benefit	The benefits of assumptions management in software development	RQ7
D16	Challenge	The challenges of assumptions management in software development	RQ7
D17	Consequence	The consequences caused by not well-managed assumptions in software development	RQ8
D18	Lesson learned	The lessons learned of assumptions management in software development	RQ9

Fig. 3. Results of search, selection, and Snowballing.

4. Results

This section reports results of the SMS.

4.1. Overview

This section first introduces the search, selection, and Snowballing results (see Section 4.1.1) and then provides the distribution of the selected studies (see Section 4.1.2).

4.1.1. Search, selection, and snowballing results

As shown in Fig. 3, during the database search, 19,428 papers were retrieved from the seven databases. 302 papers were included after the 1st round selection and 108 papers were retained after the 2nd round selection. Furthermore, 25 papers were identified through Snowballing, and one paper was selected during the extended search and selection; this led to a final set of 134 papers in this SMS.

4.1.2. Demographics of included studies

The number of the selected studies per type of venue (i.e., conference, journal, workshop, and book) is shown in Fig. 4. Most of the studies were published in conferences (81 out of 134, 60.4%%) and followed by journals (34 out of 134, 25.4%%). The 134 studies are distributed in 94 publication venues.

The number of studies per publication venue is listed in Table 18 (we only present the venues that published at least two studies). The top two venues are International Conference on Computer Aided Verification (6 out of 134, 4.5%%) and Journal of Systems and Software (5 out of 134, 3.7%%). The number of the selected studies per year ranges is from four to fifteen as shown in Fig. 5. One research group published four studies in 2008, which can partially explain the peak (i.e., 15).

Fig. 4. Number of studies over the four types of publication venues.

As shown in Fig. 6, the authors of 94 studies (out of 134, 70.1%%) were from academia, and the authors of 8 studies (out of 134, 6.0%%) came from industry. 32 studies (out of 134, 23.9%%) were jointly written by authors from both academia and industry.

4.2. RQ1: what are the definitions of assumption in software development?

Table 4 shows all the definitions of assumption in software development from the selected studies, which are further classified into a number of types and mapped to the related software development activities. Each type is explained by the corresponding definition except for two types that aggregate definitions: “Context assumption” (where

Fig. 5. Number of studies over time period (2001–2015).

Fig. 6. Number of the included studies over types of authors.

context may refer to a system environment such as hardware or the artifacts' context such as the reason of making certain decisions), and "General assumption" (defining assumption in a general way, for example, based on English dictionaries). Compared to the original definitions in the selected studies, we rephrased the sentences of the definitions taking care not to alter their meaning.

4.3. RQ2: what are the types of assumptions in software development?

In this SMS, we used and adapted the classification of software development activities proposed in SWEBOK [12] (see Appendix B), and classified assumptions according to the activities. Though there are different classifications of software development activities, the reason for choosing SWEBOK is that it is a well-accepted and mature point of reference in the domain of software development. The results are shown in Table 19: assumptions related to requirements engineering, software design, construction, testing, and maintenance and evolution. For each activity we classified assumptions either because the assumptions are made in that activity, or because the assumptions concern artifacts of that activity (e.g., requirements, architectural design decisions, source code, or bugs). Since a selected study may mention assumptions with different software development activities, the study can be classified to more than one type (see Table 19).

Specific types of assumptions in requirements engineering and software design have been further classified into various subtypes in several studies (available online [39]). Two examples are provided below.

- (1) Classification of domain assumptions [S115] (requirements engineering):
 - a Mandatory domain assumptions are assumptions about the necessary requirements in product line development (e.g., assuming a requirement should be included).

- b Optional domain assumptions are assumptions about choosing one requirement from the alternative requirements (e.g., assuming a requirement should be chosen from several requirements).
 - c Multiple domain assumptions are assumptions about choosing a number of requirements from the alternative requirements.
- (2) Classification of architectural assumptions [S84, S85] (software design):
 - a Technical assumptions are assumptions about the technical environment of the system (e.g., databases).
 - b Organizational assumptions are assumptions about the company (e.g., the social settings of the company) that may have an impact on software development.
 - c Management assumptions are assumptions about the decisions for business objectives (e.g., management strategies).

4.4. RQ3: which software artifacts are related to assumptions in software development?

Various software artifacts are related to assumptions in software development as shown in Table 5. We also classified these artifacts according to the software development activities suggested in SWEBOK [12] (see Appendix B). The reason for choosing software development activities as well as SWEBOK to classify the artifacts is the same to the classification of assumptions, which has been explained in Section 4.3. Though one type of artifacts can be involved in one or several software development activities, in this classification, only the primary activity was considered in which the type of artifacts are created and managed (e.g., requirements in requirements engineering). The reason is that RQ3 regards which software artifacts are related to assumptions, instead of relationships between software development activities and artifacts. For example, we focused on whether requirements are related to assumptions, instead of whether requirements are considered in requirements engineering or other activities in the selected studies. The artifacts were further sub-classified based on their names (e.g., "model" and "specification") as shown in Table 5. The intension is merely to improve the readability, since arranging all the artifacts into one cell would make the table difficult to read. Additionally, though both the results of RQ3 (in Table 5) and RQ2 (in Table 19) are classified based on software development activities, the two tables could not be merged. This is because the results of RQ3 only present the studies that explicitly mention assumptions with the related artifacts; in contrast, the results of RQ2 are broader and include the studies that discuss assumptions made in a specific development activity without mentioning the related artifacts. Finally, as a paper may mention assumptions with different types of artifacts, and therefore, the paper can be classified to more than one type (see Table 5).

Table 6 shows nine types of relationships between assumptions and other types of software artifacts from the studies.

We provide several representative examples to elaborate on the relationships between assumptions and software artifacts. As discussed in Section 2, the assumption concept is subjective and context-dependent. However, the following examples aim to explain the relationships, instead of distinguishing assumptions from other concepts (e.g., constraints); therefore, the examples of assumptions are provided without their detailed context.

4.4.1. Requirements engineering

In [S108], assumptions were made for requirements (i.e., A is made for B). One example of such assumptions is "The system will be available during normal working hours". In [S89], assumptions are scattered within multiple quality attribute scenarios (i.e., A is contained in B). One example of such assumption is "There is a subsystem that is responsible for receiving emergency calls and forwarding them to an available Coordinator". This assumption was defined in two quality attribute scenarios. In [S100], the authors mentioned that invalid assumptions might lead to a

Table 4
Definitions of assumption in software development.

Type	Software development activities	Core concept	Definition	Studies
Context assumption	Requirements engineering, Software design, Software maintenance and evolution	Uncertainties and expectations	Context assumptions are uncertainties and expectations of the context (e.g., motivational reasons) for making design decisions.	[S122]
	Requirements engineering	Assumptions	Context assumptions are assumptions about the laws of the physical world and the behavior of other systems.	[S98]
	Requirements engineering	Uncertain descriptive statements	Context assumptions are descriptive statements that may not hold and should be satisfied by the problem world.	[S38]
	Requirements engineering, Software design, Software construction	Assumptions	Context assumptions are assumptions that constrain environment behaviors.	[S17]
Trust assumption	Requirements engineering, Software design	Uncertain statements	Context assumptions are statements accepted to be true, and they are about the hardware and everything outside the system.	[S99]
Architectural assumption	Requirements engineering, Software design	Trusted choices, statements, and opinions	Trust assumptions are explicit or implicit choices, statements, or opinions that concern the behavior and properties of the system.	[S45][S60][S61]
	Requirements engineering, Software design, Software construction, Software testing, Software maintenance and evolution	Implicit design decisions and the rationale	Architectural assumptions are implicit design decisions as well as the rationale and context behind these decisions.	[S62][S107][S108] [S85][S88][S91] [S114][S132]
Early architectural assumption	Requirements engineering, Software design	Assumptions	Early architectural assumptions are assumptions about initial architectural elements of the expected architecture (e.g., building blocks of the envisaged system) and they exist before making architectural design decisions.	[S89][S90][S91]
General assumption	Requirements engineering, Software design, Software maintenance and evolution	Uncertain assertions	Assumptions are assertions of truth or something that is taken for granted.	[S63]
Aspect assumption	Requirements engineering, Software design, Software construction, Software testing, Software maintenance and evolution	Invariabilities	Assumptions are invariabilities about the system and the environment.	[S85]
	Software design	Assumptions	Aspect assumptions are assumptions about the context of an aspect (e.g., the property of the aspect) in aspect-oriented software development.	[S133]
Service assumption	Software design	Assumptions	Service assumptions are assumptions about the things that may not be known in service composition.	[S97]

Table 5
Software artifacts related to assumptions in software development.

Software development activity	Number (%)	Software artifact	Number (%)	Studies
Requirements engineering	72 (53.7%)	“Misc”: Requirement, Function/Functionality, Feature, Use case, Requirements decision, User feedback, Goal, Functional contract, Requirements prototype	69 (51.5%)	[S2][S3][S4][S5][S6][S7][S8][S9][S13] [S16][S17][S18] [S29][S30][S32][S34][S36][S38][S40][S43] [S45][S46] [S53][S54][S56][S59][S60][S61][S62][S64] [S66][S67] [S70][S73][S74][S76][S78][S80][S82][S84] [S85][S88] [S89][S90][S91][S93][S96][S98][S99] [S100][S102][S103] [S104][S106][S107][S108][S111][S113] [S114][S115][S116] [S119][S121][S122][S125][S126][S127] [S128][S133] [S2][S6][S43][S60][S61][S64][S66][S104] [S109][S113] [S115][S116][S119][S121][S125] [S4][S38][S67][S82][S85][S90][S96][S112] [S115][S127] [S44][S59][S88][S89][S90][S91] [S7][S8][S9][S10][S11][S12][S13][S14] [S15][S17] [S19][S20][S21][S22][S23][S25][S26][S27] [S29] [S30][S33][S34][S35][S36][S40][S41][S43] [S44][S46] [S52][S53][S54][S55][S56][S58][S60][S64] [S65][S67] [S68][S69][S70][S71][S72][S73][S75][S76] [S77][S78] [S80][S81][S83][S84][S85][S88][S89][S91] [S93] [S94][S95][S97][S99][S102][S103][S109] [S110][S111] [S114][S116][S117][S119][S120][S122] [S123][S124] [S126][S130][S131][S133][S134] [S9][S18][S24][S31][S39][S40][S59][S66] [S85][S86][S87] [S88][S89][S90][S92][S97][S99][S104] [S105][S114] [S119][S121][S122][S127][S132] [S8][S27][S31][S55][S59][S67][S68][S78] [S80][S84][S85] [S88][S89][S90][S91][S114][S119][S130] [S34][S54][S55][S71][S119][S122][S127] [S12][S19][S26][S57][S122][S134] [S57][S77][S78][S133]
Software design	96 (71.6%)	“Specification”: Requirements specification, Feature specification, Functional specification, Use case specification, User specification, Product specification, Machine specification, Domain specification, Behavior specification, Security control specification “Model”: Requirements model, Goal model, Feature model, Use case model, Domain model, Behavior model “Scenario”: Requirements scenario, Quality attribute scenario, Usage scenario “Component-Interface”: Component, COTS product, Component service, Component contract, Port, Component model, Interface, Interface event, API, API usage pattern, Component trace, Service, Module, Package	15 (11.2%) 10 (7.5%) 6 (4.5%) 80 (59.7%)	
		“Decision”: System decision, Design decision, Modeling decision	25 (18.7%)	
		“Architecture”: Architecture, Architectural view, Architectural solution	18 (13.4%)	
		“Misc”: Design model, Dataflow, Control flow, Design pattern, Design contract, Design object, Design persona “Specification”: Design specification, Architecture specification, Component specification, Service specification, Aspect specification “Aspect”: Aspect, Aspect property model	7 (5.2%) 6 (4.5%) 4 (3.0%)	

(continued on next page)

Table 5 (continued)

Software development activity	Number (%)	Software artifact	Number (%)	Studies
Software construction	31 (23.1%)	"Code": Source code, Source code comment, Source code annotation, Implementation decision, Class, Thread	31 (23.1%)	[S1][S3][S6][S13][S17][S23][S34][S36][S40][S42][S44][S47][S51][S54][S55][S57][S75][S76][S80][S86][S87][S93][S100][S101][S105][S109][S111][S114][S119][S124][S133][S93][S109]
Software testing	5 (3.7%)	"Specification": Program specification, Data values specification Bug, Testing plan	2 (1.5%)	[S13][S54][S80][S105][S133]
Others	16 (11.9%)	Risk, Organizational decision, Management decision, Plan, Task, Context model, Context specification, Version control information, System log, Process model, Product model	5 (3.7%) 16 (11.9%)	[S5][S31][S38][S39][S43][S45][S60][S67][S76][S78][S88][S100][S104][S105][S114][S124]

Table 6

Relationships between assumptions and other types of software artifacts.

Relationship	Description
A is made for B	Assumptions are made for artifacts or vice versa.
A is based on B	Assumptions are made according to artifacts or vice versa.
A is contained in B	Assumptions are part of artifacts.
A is B per se	Assumptions are artifacts per se.
A leads to the problems in B	Assumptions lead to problems in artifacts or vice versa.
A leads to the changes in B	Assumptions lead to changes in artifacts or vice versa.
A helps to manage B	Assumptions help to manage artifacts or vice versa.
A's failure negatively impacts B	Assumptions failure negatively impacts artifacts or vice versa.
A is related to B	There is a relationship between assumptions and artifacts, but the relationship type is not explicitly mentioned in the study.

change in requirements, and the changes of the operational domain could be a reason for assumptions changes (i.e., A leads to the changes in B).

4.4.2. Software design

In [S88], architectural assumptions may facilitate design decisions making (i.e., A helps to manage B). One example of such assumption is "There will be one separate subsystem for processing the incoming information updates, and that it will use a priority queue for the incoming data packages". This assumption was used to reduce the solution space and helped architects to make architectural design decisions. In [S114], the authors defined assumptions as implicit design decisions (i.e., A is B per se) as well as the rationale and context behind these decisions (i.e., A is based on B). The assumptions recovered in this study include management assumptions, organizational assumptions, and technical assumptions. In [S86], the authors pointed out that design decisions in a system are based on context assumptions (i.e., A is based on B).

4.4.3. Software construction

In [S44], assumptions were made about the source code of API clients (i.e., A is made for B). One example of such assumption is "Method M should not be called by the API clients". In [S93], the authors suggested identifying and recording assumptions as part of program specification (i.e., A is contained in B). In [S47], assumptions were used to model the behavior of threads (i.e., A helps to manage B). One example of such assumption is "If thread i holds the lock in read mode, then x cannot be changed by another thread".

4.4.4. Software testing

In [S105], the case study results show that "bug fixes" is an important reason for technical assumptions failures, and 8 out of 13 assumptions failures were caused by bug fixes (i.e., A leads to the problems in B). One example of such assumption is "Two systems do not affect each other in any way". In [S133], assumptions were considered to help in finding bugs (i.e., A helps to manage B), and certain types of assumptions should be treated as a type of bug to be fixed.

4.4.5. Others

In [S100], invalid assumptions can increase the associated risks of the system (i.e., A's failure negatively impacts B), because of, for example, the reduced validity of the related requirements. Modeling the relationships between assumptions and requirements is one way to predict system risks. In [S78], both product models and process models could contain a set of assumptions (i.e., A is contained in B). One example of such assumptions is to assume the scope of the product.

4.5. RQ4: which activities have been proposed to support assumptions management in software development?

Twelve assumptions management activities were identified and classified from the studies:

Assumptions Making (including Identification) is aimed at both making new assumptions and identifying existing assumptions in software development, and analyzing the properties of the assumptions (e.g., validity, pros, and cons).

Assumptions Description is used to describe and record assumptions in certain forms.

Assumptions Evaluation (including Satisfaction) ensures that the description and analysis of assumptions are correct and accurate, and checks whether the assumptions are satisfied by other software elements (e.g., components).

Assumptions Maintenance adapts assumptions to fit the context of software development (e.g., eliminating outdated or invalid assumptions, modifying conflicting assumptions, and transforming assumptions into other types of software artifacts).

Assumptions Tracing is used to connect assumptions to other types of software artifacts (e.g., requirements).

Assumptions Monitoring (including Evolution) reflects the changes of assumptions (e.g., the properties of assumptions) during the software development life cycle.

Assumptions Communication refers to sharing and discussion of assumptions, which can reduce misunderstandings of assumptions among different stakeholders.

Assumptions Reuse aims at reusing existing assumptions through adapting them for other contexts (e.g., other projects).

Assumptions Understanding concerns comprehending assumptions themselves as well as their relationships.

Assumptions Recovery relates to regaining the assumptions made in software development.

Assumptions Searching refers to retrieving assumptions from project resources (e.g., project documents and repositories).

Assumptions Organization concerns sorting and classifying the existing assumptions of a project.

Table 20 presents the classification of the assumptions management activities with their studies. Fig. 7 shows the distribution of the studies over the assumptions management activities (see Section 4.5), time period, and types of assumptions based on software development activities (see Section 4.3). The bubbles on the left side of the chart represent the studies on specific assumptions management activities over publication year, while the bubbles on the right side show the studies on certain types of assumptions over publication year. The numbers in each bubble denote the studies.

4.6. RQ5: which approaches and tools are available to support assumptions management in software development?

4.6.1. Approaches

As shown in Table 21, we collected the approaches of assumptions management and classified them according to the corresponding assumptions management activities (see Section 4.5). Note that not every assumptions management activity has been supported by certain approaches, and one approach can be used to support one or multiple assumptions management activities. Also, we only classified an approach to an assumptions management activity if the study explicitly mentioned the relationship between the approach and the activity. For example, Assumptions Maintenance is a common step in assume-guarantee reasoning. However, some of the studies related to assume-guarantee reasoning did not explicitly mention this step (i.e., Assumptions Maintenance). Therefore, such studies were not classified into Assumptions Maintenance. Next we provide some approaches for each assumptions management activity, as examples.

4.6.1.1. Assumptions Making and Evaluation. According to the results of the SMS, the representative approach of Assumptions Making and Evaluation is assume-guarantee reasoning [37,38]. Considering there is a need to evaluate whether a system satisfies a property, instead of verifying the whole system, assume-guarantee reasoning decomposes the system into components (i.e., which are smaller and easier to verify), generates assumptions for a component that guarantees the property, checks whether the other components satisfy the assumption, and then comes up with the results. There are different variants of assume-guarantee reasoning; here we provide two examples. In [S14], the authors pointed out that the general assume-guarantee rules are: (a) decomposing a system into two components M1 and M2; (b) checking that M1 under an assumption satisfies the property; and (c) discharging the assumption on the context of M2. If the results are true, then the system satisfies the property. In [S109], the authors described the process of assume-guarantee reasoning as: (a) making an assumption of the context of the component, which characterizes the expected behavior of the component; (b) constructing the environment of the component based on the assumption; and (c) checking the system to evaluate if the component satisfies its behavior specification based on the assumption with no errors.

4.6.1.2. Assumptions Description. There are numerous types of Assumptions Description approaches, such as using documents (e.g., Model Sequence Diagrams specifications), languages (e.g., a property-based language), frameworks and models (e.g., with templates), graphs, interfaces, contracts, personas, arguments, and analogy. In [S84], the authors integrated Assumptions Description as part of a component template, including components assumptions and interfaces assumptions. In [S97], the authors used Semantic Web Rule Language to express assumptions. In [S36], the authors proposed an Assumptions Description approach for constructing documents using natural language, showing how assumptions and requirements are related, and which assumptions are unused.

4.6.1.3. Assumptions Maintenance. In some assume-guarantee reasoning approaches (e.g., [S14, S29, S81]), there is a step to refine the unsatisfied assumptions after Assumptions Evaluation. In [S97], the authors proposed a framework, which can be used to maintain assumptions (e.g., modifying the outdated assumptions). In [S10], the authors developed a procedure, which is composed of four steps to maintain assumptions in the context of component interfaces adaption.

4.6.1.4. Assumptions Tracing. Most of the approaches (4 out of 5, 80%) related to Assumptions Tracing support the tracing between assumptions and requirements. In [S36], the authors traced assumptions to requirements and source code, and the proposed approach can show which requirements can be related to assumptions. In [S60], the authors also traced assumptions to requirements. Specifically, they focused on the relationships between trust assumptions and security requirements. In [S96], the authors proposed to use a goal model to trace assumptions to requirements. The goal model is composed of three parts: the goals, the domain assumptions, and the relationships between them.

4.6.1.5. Assumptions Monitoring. In [S4], the authors proposed an approach to monitor assumptions (e.g., the evolution of assumptions) in a requirement model at runtime, and they used goals and goal models in their approach. They defined requirements monitoring as “keeping track of the execution of each software variants and the impact it has on requirements”. In [S104], the authors discussed the usefulness of monitoring assumptions (e.g., Assumptions Monitoring can be used to evaluate assumptions at runtime). They embedded Assumptions Monitoring as part of their proposed approach. For example, the approach can generate logs of the system through Assumptions Monitoring.

Fig. 7. Bubble chart over the assumptions management activities, time period, and types of assumptions (the numbers in each bubble denote the studies, see Appendix A).

4.6.1.6. Assumptions Recovery. For Assumptions Recovery, the approach proposed in [S114] employed five sources: free interviews, financial reports, documentation, version control, and source code to detect suspicious effects, and then identifying assumptions through guided interviews.

4.6.2. Tools

34 tools were collected from the studies, and over half of the tools (19 out of 34, 55.9%) were used to support assume-guarantee reasoning. We classified the tools into two types (i.e., “Assume-guarantee reasoning” and “Others”) as listed in Tables 7 and 8, respectively. The results shown in the “URL”, “Developer”, and “Open source” columns were not collected from the primary studies but from the Internet. “Not mentioned” in Tables 7 and 8 means that we could not find evidence regarding, for example, whether a tool is open source or not.

4.6.2.1. Assume-guarantee Reasoning. The tools of this type were used for assume-guarantee reasoning in software development. For example, in [S110], the authors implemented LTSA (Labeled Transition System Analyzer), which supports their proposed assume-guarantee reasoning approach (e.g., the assumptions learning framework).

4.6.2.2. Others. The tools of this type can be used to support specific assumptions management activities in software development. In [S122], the AREL tool was developed to support design rationale

identification and tracing to other elements, including design decisions, assumptions, and constraints. In [S7], the OCRA tool was mainly used for component-based design, but can also support Assumptions Description.

4.7. RQ6: which stakeholders are involved in assumptions management in software development?

As shown in Table 9, we collected stakeholders from the studies and classified them according to the software development activities. The reason for choosing software development activities as well as SWEBOK to classify the stakeholders is the same to the reason for the classification of assumptions, which has been explained in Section 4.3.

Besides relating the identified stakeholders to assumptions management activities, there are four other types of involvement between stakeholders and assumptions management.

(1) Stakeholders impact or are impacted by assumptions management

In [S3], if the information used by requirements engineers is not confirmed with customers, the requirements engineers have to make assumptions. These assumptions can lead to many problems such as rework and project failures. In [S122], architects would have trouble in architecture evaluation and maintenance, if assumptions were implicit.

(2) Stakeholders use assumptions for specific activities in software development

Table 7
Tools for assume-guarantee reasoning.

Tool	URL	Developer	Open source	Studies
LTSA (Labelled Transition System Analyzer)	http://www.doc.ic.ac.uk/ltsa/	Academia	Not mentioned	[S29][S41][S53][S71][S72][S95][S110]
NuSMV model checker	http://nusmv.fbk.eu/	Academia	Yes	[S22][S26][S77]
Java PathFinder	http://javapathfinder.sourceforge.net/	Industry	Yes	[S55][S109]
SPIN model checker	http://spinroot.com/spin/whatispin.html	Academia	Yes	[S35]
AG tool	Not mentioned	Academia	Not mentioned	[S71]
AR tool	Not mentioned	Academia	Not mentioned	[S71]
SpaceEx (a symbolic hybrid model checker)	http://spaceex.imag.fr/	Academia	Yes	[S15]
BEG (Bandera Environment Generator)New name: Open Components and SystemsEnvironment Generator (OCSEGen)	https://forge.cis.ksu.edu/projects/beg/ https://code.google.com/archive/p/envgen/	Academia	Yes	[S124]
BPC Checker (Behavior Protocol Model Checker)	Not mentioned	Not mentioned	Not mentioned	[S109]
EnvGen (Environment Generator for Java PathFinder)	Not mentioned	Academia	Yes	[S109]
SPARK (a prototype tool in Python)	http://www.altran.co.uk/	Industry	Not mentioned	[S76]
SGM (State-Graph Manipulators)	https://www.cs.ccu.edu.tw/~pahsiung/sgm/ http://embedded.cs.ccu.edu.tw/~esl_web/Project/Ch/SGM/ http://vedder.se/SP/SafetyAddDraft.pdf	Academia	Not mentioned	[S126]
SafetyADD	Not mentioned	Not mentioned	Not mentioned	[S30]
LUTE checker	Not mentioned	Not mentioned	Not mentioned	[S30]
AGREE	Not mentioned	Not mentioned	Not mentioned	[S30]
KIND model checker	http://clc.cs.uiowa.edu/Kind/	Both	Yes	[S30]
An automatic checking tool	Not mentioned	Academia	Not mentioned	[S47]
Cadence SMV	http://www.kenmcmil.com/smv.html	Academia	Yes	[S58]
SAT solver	Not mentioned	Industry	Not mentioned	[S58]

In [S4], designers evaluate assumptions and use the identified invalid assumptions to revise the related requirements models. In [S55], since the actual environment of the system is unknown during system verification, developers have to use environment assumptions to describe system properties.

(3) The content of assumptions is related to stakeholders

In [S46], the authors pointed out that some assumptions were about the profiles of users. In [S78], the authors mentioned that some assumptions are about user behaviors.

(4) General relationships between assumptions and stakeholders

In [S101], the authors argued that there was a need for user involvement in managing assumptions. In [S105], the authors described that developers need to be careful about the assumptions made outside their working context.

4.8. RQ7: what are the benefits and challenges of assumptions management in software development?

The benefits were classified into “Benefits for software development activities” and “Other benefits” as presented in Section 4.8.1 and 4.8.2, respectively.

4.8.1. Benefits for software development activities

As shown in Table 10, we further classified “Benefits for software development activities” into “Requirements engineering”, “Software design”, “Software construction”, “Software testing”, and “Software maintenance and evolution” according to SWEBOK [12] (see Appendix B).

4.8.2. Other benefits

As shown in Table 11, we further derived various types of “Other benefits” that are not related to specific software development activities.

Table 8
Other tools for assumptions management.

Tool	Assumptions management activities	URL	Developer	Open source	Studies
Alloy analyzer	AD, AE	http://alloy.mit.edu/alloy/	Academia	Not mentioned	[S67][S101][S116]
OCRA (Othello Contracts Refinement Analysis)	AD	https://es-static.fbk.eu/tools/ocra/	Academia	Not mentioned	[S7][S25]
A prototype tool	AM, AD	http://people.cs.kuleuven.be/~dimitri.vanlanduyt/ea/	Academia	Yes	[S90]
Drools (a Business Rules Management System)	AM, AD, AE	https://www.drools.org/	Industry	Yes	[S51]
PicoSAT (an SAT solver)	AM, AD, AE	http://fmv.jku.at/picosat/	Academia	Yes	[S51]
Microsoft threat modeling tool	AD	https://www.microsoft.com/en-us/download/details.aspx?id=49168	Industry	Not mentioned	[S119]
Redhat's RPM	AD	https://www.redhat.com/en	Industry	Yes	[S119]
Debian's APT	AD	https://www.debian.org/index.en.html	Industry	Yes	[S119]
XML-based engine (a tool with an XML back-end)	AD, AE	Not mentioned	Academia	Not mentioned	[S123]
PAT (Process Analysis Toolset)	AD, AE	Not mentioned	Not mentioned	Not mentioned	[S120]
SCENARIOTOOLS (an Eclipse-based tool)	AD	http://scenariotools.org/	Academia	Yes	[S16]
GASR (General-purpose Aspectual Source Code Reasoner)	AE	https://github.com/cderoove/damp.ekeko.aspectj	Academia	Yes	[S42]
AREL (Architecture Rationale and Elements Linkage)	AT	http://www.ict.swin.edu.au/personal/atang/AREL-Tool.zip	Academia	Not mentioned	[S122]
MAVEN (Modular Aspect Verification)	AD, AE	Not mentioned	Academia	Not mentioned	[S57]
CAIRIS (Computer Aided Integration of Requirements and Information Security)	Not mentioned	https://github.com/failys/CAIRIS	Academia	Yes	[S43]

Table 9
Stakeholders involved in assumptions management.

Software development activity	Stakeholder	Assumptions management activities	Studies
Requirements engineering	Customer, Client, User, System administrator	Assumptions Making, Description, Evaluation, Maintenance, Communication, Understanding	[S2][S3][S17][S38][S43][S46][S78][S79][S80][S91][S92][S93][S96][S97][S99][S100][S101][S104][S105][S113][S119][S129][S132]
	Requirements engineer	Assumptions Making, Description, Evaluation, Communication, Understanding	[S3][S4][S32][S48][S60][S61][S62][S63][S67][S79][S88][S89][S90][S91][S104][S112][S116][S121][S127]
	Domain expert	Assumptions Making, Evaluation, Communication	[S61][S101][S115][S116]
Software design	Designer	Assumptions Making, Description, Evaluation, Maintenance, Monitoring, Communication, Understanding	[S4][S5][S7][S12][S18][S23][S24][S34][S59][S68][S86][S96][S104][S116]
	Architect, Modeler, Component designer, Component user, Design decision maker	Assumptions Making, Description, Evaluation, Maintenance, Tracing, Communication, Recovery, Understanding	[S35][S59][S67][S75][S88][S89][S90][S91][S99][S114][S121][S122][S124][S132]
Software construction	Developer, Programmer	Assumptions Making, Description, Evaluation, Tracing, Communication, Recovery, Organization	[S2][S6][S12][S39][S40][S47][S50][S53][S54][S55][S74][S77][S78][S86][S100][S105][S107][S112][S113][S114][S116][S119][S124][S127][S129]
	API developer, API user, Component developer, Aspect developer, Library user, System integrator, Implementation consultant	Assumptions Making, Description, Evaluation, Maintenance	[S17][S44][S93][S123][S132][S133]
Software maintenance and evolution	Maintainer	Assumptions Making	[S78][S86]
Others	Project manager, Deployment manager	Assumptions Making	[S78][S114][S132]
	Technical stakeholder, Technical expert	Assumptions Making, Evaluation	[S76][S88][S91]

Table 10
Benefits of assumptions management related to software development activities.

Software development activity	Benefits	Studies
Requirements engineering	(1) Explicating tacit knowledge of users through tracing assumptions to their source, (2) explicating the claims that the assumptions represent and helping requirements analysis, description, and verification through formally modeling assumptions, requirements, and behaviors of system, (3) helping to assess the quality and completeness of requirements specifications, (4) helping to understand the perceptions of requirements engineers, (5) saving time in requirements engineering, (6) helping consistency management in requirements engineering, (7) reducing the problem space, (8) enabling the derivation and exploration of interrelations between different requirements, (9) helping to identify requirements, (10) being aware of requirements gaps, and (11) helping to build prototypes.	[S2][S3][S5][S16][S18][S32][S38][S43][S59][S74][S85][S88][S89][S96][S113][S115][S121]
Software design	(1) Supporting service composition through using service assumptions, (2) helping to understand architecture design through relating and reasoning requirements, assumptions, constraints, and design decisions, (3) helping aspect design and reuse, (4) helping to identify architectural interaction points, (5) improving the effectiveness of conducting architectural activities, (6) easing the transition from requirements engineering to architectural design, (7) reducing the effort required for handling models, (8) supporting architectural knowledge management, (9) allowing reasoning with incomplete information in software design, and (10) improving the quality of models and architecture.	[S5][S19][S34][S35][S53][S59][S70][S77][S83][S85][S89][S94][S96][S97][S109][S110][S114][S119][S122][S123][S127][S130][S132][S133]
Software construction	(1) Allowing for checking that safety properties are preserved through design-level assumptions, (2) improving the quality of source code, (3) helping in understanding, evaluating, and constructing source code; ensuring information-flow security through using assumptions at thread level.	[S1][S19][S53][S54][S55][S86][S87][S111][S133]
Software testing	(1) Helping to reduce testing time through using assume-guarantee properties in component testing, and (2) bugs identification and fixing.	[S11][S13][S54][S123][S133]
Software maintenance and evolution	(1) Effectively rechecking component-based systems based on components evolution through using the generated minimized assumptions, (2) helping discussions about the further evolution of the system, (3) improving knowledge transfer to support software maintenance and evolution, (4) making software evolution reliable, (5) making software maintenance simple, (6) assisting evolution release planning, (7) helping to detect, reduce, and fix, for instance, system failures/errors, end-product defects, system-level violations, and vulnerabilities through for example, making assumptions explicit, (8) providing warnings about application and system mismatches, and (9) reducing the risks of, for example, system failures.	[S5][S13][S34][S35][S45][S54][S61][S62][S71][S72][S84][S85][S86][S87][S92][S93][S104][S111][S116][S123][S114][S115][S133]

Table 11
Other benefits of assumptions management in software development.

Benefit type	Benefits	Studies
System verification	(1) Facilitating verification of software systems, including modular verification, component verification, and program verification, through, for example, providing automatic modular verification based on assume-guarantee reasoning, (2) helping effective model checking of open systems and synthesis, and (3) enhancing the efficiency of model checking (e.g., performance) of source code.	[S14][S20][S21][S35][S38][S41][S47][S55][S65][S70][S72][S76][S83][S110][S120]
Cost	(1) Reducing computational cost for modular verification in component-based systems through using assume-guarantee techniques, and cost of the maintenance of the design through prototyping assumptions with requirements, (2) helping to assess the cost of changing the system, (3) saving money, and (4) avoiding costly mistakes.	[S5][S34][S35][S54][S56][S63][S71][S72][S92][S114][S115]
Traceability	(1) Improving the traceability through tracing assumptions to software artifacts, and (2) helping to trace assumptions to design and implementation, and vice versa.	[S36][S45][S54][S85][S105][S111][S116][S122][S127]
Software development	(1) Improving API development (e.g., automatically checking whether the usage of an API is well-behaved) through expressing assumptions as syntactic constraints, (2) improving embedded system development (e.g., helping modeling and analysis) through using a model based assume-guarantee principles, (3) improving multithreaded programs development (e.g., helping to model the behaviors of threads) through using assumptions and guarantees, (4) complementing agile development, (5) providing guidance for development process, (6) helping to design new software processes, (7) making software development and the usage of software reliable, (8) smoothening the transition between successive development iterations in iterative development, and (9) reducing the effort of software development.	[S6][S44][S47][S65][S80][S91][S92][S105]
Inconsistency management	(1) Detecting mismatches between requirements and design decisions through using derivational analogy, (2) minimizing the risk that assumptions are incompatible with the specifications that are embedded in the assignment.	[S19][S36][S59][S75][S93][S97][S115]
Stakeholder	(1) Helping users and designers to have a clear and consistent understanding on the software through using the structure and goal models that trace assumptions to requirements, (2) helping software teams to think more carefully not only about the application's design but also how assumptions are manifested in software architectures, testing plans, and triage processes, (3) helping stakeholders to set more realistic expectations, (4) helping stakeholders to understand the modeling activities and the system, (5) giving stakeholders more structure to the way of thinking and argumentation, and (6) gaining knowledge in the early phase of the development through integrating and executing the software models with assumptions in a virtual environment.	[S38][S40][S80][S85][S96][S99][S133]
Software quality	(1) Improving software quality through, for example, making different types of assumptions of components explicit and making conscious decisions about the system's inflexibility with respect to those assumed invariabilities, (2) improving system's safety and reliability, and (3) enabling web services applications to be more flexible and intelligent.	[S75][S85][S92][S97][S100][S123][S76][S93][S97]
Miscellaneous	(1) Helping to identify runtime alternatives for security control specifications that are more effective, (2) helping the satisfaction of security requirements, (3) using non-determinism in Assumptions Description can facilitate system configuration through allowing analyzing the configuration-independent properties without enumerating the combinations of configuration settings, (4) helping runtime monitoring and component retrieval through generating the weakest environment assumptions, (5) improving the description of system (e.g., system behaviors) through, for instance, using an extension of the play-out algorithm with environment assumptions, (6) making it easier to reproduce problematic behaviors through checking components in isolation using assume-guarantee properties in component testing, (7) reducing the risks in projects (e.g., managing the risks of invalid assumptions through the partial automation process and analyzing risks through using assumptions as an explicit target), (8) helping bridge the gap between a system and its operational environment, (9) reducing unexpected or strange behavior in execution, increasing the confidence in system operation, and improving consistency management in execution	[S5][S16][S53][S54][S60][S67][S99][S104][S124]

4.8.3. Challenges

The challenges of assumptions management in software development were classified into two types, i.e., “*Assumption*” (as an artifact) and “*Assumptions management*” (as an activity) with twelve subtypes as presented in Table 12.

4.9. RQ8: what are the consequences when assumptions are not well managed in software development?

We classified the consequences when assumptions are not well managed into five types (i.e., “*Invalid assumption*”, “*Implicit assumption*”, “*Inconsistent assumption*”, “*Badly modularized assumption*”, and “*Others*”) with twenty-six subtypes as presented in Table 13. “*Invalid assumption*” includes invalid assumptions, violated assumptions, and failed assumptions.

4.10. RQ9: what are the lessons learned from assumptions management in software development?

Normally a lesson learned can be presented in one sentence, several sentences, or several paragraphs, and there are more than 400 lessons learned identified from the studies. Therefore, we did not list all the lessons learned in this paper due to the space limitation, but made them available online [39]. We further classified them into three types as the following:

(1) Impact of assumptions

This type refers to the lessons learned concerning how assumptions impact software development, such as software artifacts, software qualities, resources, and development activities.

(2) Suggestion for assumptions management

This type covers the suggestions and guidelines proposed in the studies about assumptions management in software development (i.e., how to manage assumptions), such as assumptions classification, assumptions management and agile development, assumptions awareness, and assumptions management activities.

Table 12
Challenges of assumptions management in software development.

Challenge type	Challenge subtype	Description	Studies
Assumption	Distinguishing assumptions from other artifacts	It is difficult to distinguish assumptions from other types of software artifacts (e.g., requirements).	[S88][S90][S114][S122]
	Assumptions classification	It is difficult to classify assumptions in software development.	[S11][S85][S99]
	Software failures and invalid assumptions	It is difficult to analyze which software failures are caused by invalid assumptions.	[S92]
Assumptions management	Gap between the expected and real world	Assumptions in the expected world may not be present in the real world.	[S18]
	Assumptions management activity	It is difficult to conduct the assumptions management activities, because of, for example, the required effort, incomplete information, or the lack of approaches, tools with guidelines to support the assumptions management activities.	[S6][S8][S9][S19][S24][S34][S35][S36][S37][S38][S39][S40][S43][S44][S45][S60][S61][S66][S67][S68][S75][S76][S82][S84][S85][S86][S88][S89][S90][S91][S92][S99][S100][S104][S105][S109][S113][S114][S115][S118][S119][S122][S123][S125][S129][S130][S132][S133]
	Assumptions explosion	It is difficult to manage a large number of assumptions or assumptions with many states in the context of assume-guarantee reasoning.	[S85][S100][S102][S103][S105][S123][S134]
	Awareness	Stakeholders are not aware of the assumptions made in their work.	[S2][S92][S105][S119][S133]
	Experience and knowledge	Stakeholders do not have the required experience and knowledge for assumptions management in software development, including the unfamiliarity of the assumption term and concept.	[S43][S114][S131]
	Integration in software development	It is difficult to integrate assumptions management with software development activities or the whole software development process.	[S85][S91][S105]
	Lack of evidence of the benefits	There is a lack of evidence for the benefits of assumptions management in software development.	[S105]
	Short-term benefits	Assumptions management does not have short-term benefits.	[S100]

(3) Understanding of assumptions

This type collects the lessons learned about the understanding of assumptions in software development from the studies, such as the role of assumptions and characteristics of assumptions in software development.

Table 14 shows the studies of the three types as well as the representative examples of each type.

5. Discussion

The studies are distributed over 94 publication venues. This indicates that assumptions and their management have been a widespread topic in software engineering. Interestingly, the top two venues listed in Table 18 include only six and five selected studies respectively, and there are no venues that specifically publish studies on assumptions and their management in software development. This section presents analysis of the results (see Section 5.1), a comparison between automatic and manual assumptions management (see Section 5.2), and discusses implications for researchers (see Section 5.3) and practitioners (see Section 5.4).

5.1. Analysis of results

5.1.1. Definitions of assumption

As shown in Table 4, only twenty-one studies (out of 134, 15.7%) explicitly defined the assumption concept. One potential reason for this result is that the word “*assumption*” is general and frequently used in software development; thus the concept and the term “*assumption*” may be expected as common knowledge [13,14]. Comparing the twelve definitions from Table 4 (as shown in Table 15), most of them (5 out of 12, 41.7%) do not define the word “*assumption*”. Instead they directly use the word “*assumption*” in their definitions, and focus on what exactly assumptions concern. For example, in [S98], the authors limit the scope of context assumption as “*the laws of the physical world and the behavior of other systems*”. Four definitions (out of 12, 33.3%) explicitly mention that the essence of an assumption is “*uncertainty*”, which is consistent with the discussion in Section 1 and 2.

5.1.2. Types of assumptions and related software artifacts

As shown in Tables 5 and 19, the majority of assumptions in software development are made or related to the artifacts in requirements engineering (76 out of 134, 56.7%) and software design (104 out of 134, 77.6%). There are two potential reasons: (a) Assumptions mainly emerge in the early phases of software development (e.g., requirements engineering), due to the lack of information, knowledge, and evidence in those phases. (b) The usage of the terms in various phases could be different. For example, developers may use a different term when talking about assumptions in later development phases.

Though one type of artifacts can be involved in one or several software development activities, in this SMS, only the primary activity was considered in which the type of artifacts are created and managed (e.g., requirements in requirements engineering). As a result, no artifacts are related to assumptions in software maintenance and evolution as presented in Table 5, and consequently, this could be another reason why there are not many studies being classified as software maintenance and evolution in Table 19 (types of assumptions related to software development activities).

Furthermore, according to Tables 5 and 6, assumptions and certain type of software artifacts are interweaved in software development and are characterized by complex relationships (e.g., the relationships between an assumption and another type of artifact can be zero, one, or multiple, as well as unidirectional or bi-directional [24]). Examples and further discussion can be found in Section 4.3 and Section 5.3, respectively.

5.1.3. Assumptions management activities, approaches, and tools

As shown in Table 20, there are twelve assumptions management activities identified from the studies, but most effort has been put on Assumptions Making (108 out of 134, 80.6%), Description (89 out of 134, 66.4%), and Evaluation (83 out of 134, 61.9%). Assumptions Maintenance (30 out of 134, 22.4%) received moderate attention. Other assumptions management activities were mentioned in less than 15.0% of the studies. One reason may be that Assumptions Making, Description, Evaluation, and Maintenance were considered as primary assumptions management activities in software development; if

Table 13
Consequences when assumptions are not well managed in software development.

Consequence type	Consequence subtype	Studies
Invalid assumption	System issues (e.g., system vulnerability, system failures/errors, and integration defects)	[S6][S11][S34][S38][S39][S40][S78][S86][S87][S93][S100][S104][S116][S123]
	Negative impacts on requirements (e.g., meaningless results from requirements analysis, obstacles to the security requirements, unable to meet the requirements, system quality problems, reducing the validity of the requirements, violation of requirements, and requirements risks)	[S34][S38][S46][S67][S93][S100][S104][S108]
	Undesirable consequence (e.g., unwanted variations in product family, unwanted side-effects of defects, unsatisfied execution, and unpredictable and/or unacceptable program/system/environment behavior)	[S38][S40][S44][S84][S92][S93][S104]
	Increased cost (e.g., increased maintenance cost and extra resources and rework)	[S24][S44][S93]
	Business problems (e.g., customer dissatisfaction, loss of market share and reputation, and increased cost of business operations)	[S93][S100]
	Increased mismatch with the context of software operation	[S93]
	Increased possibility of bugs	[S44]
	Misunderstandings between API clients and providers	[S44]
	System issues (e.g., system vulnerability, system failures/errors, runtime errors, and validity problems)	[S2][S3][S6][S11][S78][S84][S86][S87][S105][S116][S119][S123][S133]
	Negative impacts to requirements (e.g., violating the requirements, false requirements, unable to meet the requirements, incomplete and not understandable requirements, requirements risks, increased uncertainty of requirements, and system quality problems)	[S2][S3][S16][S32][S40][S60][S89][S92][S121][S132][S133]
Implicit assumption	Design-related problems (e.g., design violations, component composition problems, inflexible architecture, decreased evolvability, variability, and reusability of architecture solutions, architectural misunderstanding, and risks in architecting)	[S85][S88][S89][S90][S114][S119][S122][S132]
	Increased cost (e.g., the change turns out to be very costly, the next release takes too much time, rework, expensive maintenance, the costs for fixing the issues caused by implicit assumptions)	[S2][S3][S40][S85][S89][S90][S114]
	Inconsistency problems (e.g., conflicting assumptions that lead to system mismatch)	[S32][S84][S90][S114]
	Hindering software evolution (e.g., not being aware of the changes during evolution)	[S85][S114]
	Undesirable consequence (e.g., unexpected environment behavior)	[S38]
	Traceability problems (e.g., loss of traces)	[S114]
	Project delay	[S132]
	Architectural mismatch (e.g., mismatch between components and mismatch between models)	[S49][S78]
	System issues (e.g., system failures and integration defects)	[S123]
	Project risks	[S78]
Inconsistent assumption	Interoperability conflicts between components	[S12]
	Budget and schedule overruns	[S12]
	Hindering architecture change impact analysis and trade-off analysis	[S90]
Badly modularized assumption	Increased cost (e.g., effort for understanding and analyzing requirements)	[S90]
Others	Design-related problems (e.g., expensive reconstruction of design rationale, unclear design criteria and environmental factors that impact the architecture, violating design integrity, and misunderstandings of design decisions)	[S122]
	Negative impacts to requirements (e.g., ignoring business goals and constraints, and inappropriate change impact analysis on requirements and environmental factors)	[S122]

assumptions management is undertaken, these primary activities are most likely to take place. In contrast, other activities were treated as activities of secondary significance (e.g., are not likely to happen). We provide further argumentation about the value of the four primary assumptions management activities in Table 16.

For assumptions management approaches, the situation is similar to assumptions management activities, as the top four types are Assumptions Making, Description, Evaluation, and Maintenance approaches (see Table 21). On the contrary, there are only a few approaches related to Assumptions Tracing (5 out of 134, 3.7%), Monitoring (3 out of 134, 2.2%), and Recovery (1 out of 134, 0.7%), and there are no approaches that specifically support Assumptions Communication, Reuse, Understanding, Searching, and Organization. Similarly as before, this could be attributed to not considering these activities as primary activities for assumptions management in software development.

As shown in Table 7, most of the tools (19 out of 34, 55.9%) identified from the studies were used to support assume-guarantee reasoning. One reason is that many proposed approaches from the studies are related to assume-guarantee reasoning. Moreover, as shown in Table 8, concerning the other tools (15 out of 34, 44.1%), most of them can be used to support Assumptions Description (12 out of 15, 80.0%). This is an intuitive result since stakeholders cannot manage assumptions, if they are not explicitly described.

Furthermore, the approaches and tools are related to their context of use. For example, assume-guarantee reasoning is a powerful approach for system verification, including verification of components, source code, threads, and models, and can support assumptions activities, such as Assumptions Making, Description, and Evaluation. However, in other contexts of software development (e.g., requirements analysis and design decisions making), assume-guarantee reasoning may not be suitable. Such context of use is an important factor for adopting the approaches and tools in practice.

5.1.4. Stakeholders

Table 9 shows the stakeholders involved in assumptions management activities according to the software development activities. An interesting result is that all the identified types of stakeholders are involved in Assumptions Making in their work, followed by Evaluation and Description. Stakeholders in requirements engineering (e.g., requirements engineer), software design (e.g., designer), and software construction (e.g., developer) play a central role in assumptions management. On the contrary, there are no studies that explicitly mention which stakeholders should be involved in Assumptions Reuse, Searching, and Organization.

5.1.5. Benefits and challenges

The benefits of assumptions management were classified into

Table 14
Lessons learned of assumptions management in software development.

Lessons learned type	Description	Studies
Impact of assumptions	Three examples: (1) Assumptions are part of the program properties and assumption failures impact program behavior in an unexpected fashion [S93]. (2) The authors learned that design decisions are based on the assumed environmental characteristics (i.e., assumptions) [S86]. (3) A requirement model contains assumptions, and these assumptions are the driver for requirement evolution [S4].	[S4][S5][S6][S16][S17][S18][S31][S32][S34][S35][S38][S39][S40][S43][S50][S59][S60][S61][S63][S75][S78][S80][S82][S85][S86][S88][S89][S90][S91][S92][S93][S94][S96][S99][S100][S104][S105][S106][S111][S114][S115][S116][S118][S120][S122][S133]
Suggestion for assumptions management	Three examples: (1) Based on the results of their case study, the authors concluded that the assumptions considered to be more stable by stakeholders can have a more significant impact when they turn to be invalid than those assumptions regarded less stable [S105]. (2) When constructing an interface, it is necessary to make sure that there exists at least one environment that satisfies its input assumptions [S33]. (3) The ability to model environment assumptions (e.g., the assumptions made for physical environment of the system) is crucial [S16].	[S2][S3][S4][S5][S6][S7][S8][S9][S11][S15][S16][S19][S20][S24][S26][S28][S29][S31][S33][S34][S35][S36][S38][S39][S40][S41][S42][S43][S46][S49][S53][S54][S55][S59][S60][S61][S62][S63][S65][S67][S68][S71][S72][S73][S74][S75][S76][S78][S79][S80][S84][S85][S86][S88][S89][S90][S91][S92][S93][S94][S95][S97][S99][S100][S101][S102][S103][S104][S105][S108][S109][S111][S113][S114][S115][S116][S117][S118][S119][S121][S122][S124][S126][S127][S129][S132][S133]
Understanding of assumptions	Three examples: (1) Implicit assumptions are cross-cutting, because their impacts can be diffuse [S85]. (2) Assumptions made about the environment of system operation can become invalid during system operation [S104]. (3) Different stakeholders may have different assumptions, and these assumptions can be either consistent or inconsistent [S78].	[S2][S3][S4][S5][S8][S12][S13][S18][S24][S26][S32][S34][S39][S40][S44][S45][S46][S48][S60][S63][S67][S68][S74][S75][S78][S85][S86][S88][S89][S90][S91][S92][S93][S97][S98][S100][S104][S105][S106][S114][S115][S120][S132][S133]

Table 15
Number of the definitions of assumption over core concepts.

Core concept	Number of definitions (%)
Assumption regarding a specific context	5 (41.7%)
Uncertainty	4 (33.3%)
Trusted choices, statements, and opinions	1 (8.3%)
Implicit design decisions and the rationale	1 (8.3%)
Invariabilities	1 (8.3%)

thirteen types (see Tables 10 and 11). Notably, these benefits may not come for free, because they depend on various conditions (e.g., using specific assumptions management approaches, activities, or tools), or may raise certain challenges in assumptions management. For example, documenting assumptions can make assumptions explicit which

alleviates the problems caused by implicit assumptions in software development. However, this may raise the challenge of how to document assumptions in a cost-efficient way: stakeholders may need to spend much effort on Assumptions Description, but the documents of assumptions may be unnecessary and useless in projects. Therefore, trade-offs between getting the benefits of managing assumptions and dealing with the related challenges need to be considered.

Table 12 shows the classification of the challenges of assumptions management in software development. The major challenge identified from the studies is the difficulty of performing assumptions management activities in software development. The reasons are that these activities required non-trivial effort, they were conducted with incomplete information, and there was a lack of supporting approaches and tools with guidelines for the assumptions management activities.

Table 16
Rationale of the primary assumptions management activities.

Activity	Rationale
Making	Assumptions Making is the first step of assumptions management, since no assumption can exist without making it.
Description	Despite being an important type of architectural knowledge, assumptions are usually kept undocumented in projects [11]; this is one of the most critical problems in software development, and leads to a multitude of issues (see Section 4.9). Therefore, documenting assumptions in an explicit way is of paramount significance in software development.
Evaluation	Assumptions may be invalid in the first place (see Section 2). However, this invalidity may be undetected. Moreover, assumptions have a dynamic nature, referring to that assumptions can evolve over time (see Section 2). Finally, assumptions are context-dependent, i.e., they can be different depending on the context (e.g., project context) (see Section 2). As invalid assumptions could be rather harmful (e.g., leading to architectural mismatch) in projects (see Section 4.9), Assumptions Evaluation is a necessary step to identify such invalid assumptions.
Maintenance	As mentioned in Section 2, assumptions have a dynamic nature and they are context-dependent, which can lead to a number of invalid assumptions in projects, and further cause a multitude of issues, such as architectural mismatch. This motivates the need of Assumptions Maintenance. For example, when the context changes, an assumption may need to be adapted in order to fit the new context.

5.1.6. Not well-managed assumptions

Table 13 shows the classification of the consequences when assumptions are not well managed. Most of the negative consequences are caused by invalid or implicit assumptions. One reason is that assumptions exist in different phases of software development, interact with various software artifacts, and are related to many stakeholders. If assumptions are invalid or implicit, certain aspects of the project can be negatively impacted. This result also supports the significance of assumptions in software development, and that making assumptions explicit and reducing the number of invalid assumptions are critical to mitigating the negative consequences.

5.1.7. Lessons learned

We classified the lessons learned into three types, and provided three examples for each type as shown in Table 14. Similar to the approaches and tools, the lessons learned also rely heavily on their context of use. Taking one of these lessons learned and adopting it as-is in a different context may not be an optimal fit but instead may cause significant problems in assumptions management. For example, in [S104], the authors learnt that assumptions made about the operating environment may become invalid during system operation; the context of this lesson learned is the operating environment. However, other types of assumptions, such as assumptions about components, may not become invalid during system operation.

5.2. Comparison of automatic and manual assumptions management

As discussed in Point (4) of Section 2, there is a significant difference regarding automatic (AAM) and manual (MAM) assumptions management. In this section, we compare and discuss AAM and MAM regarding thirteen different aspects based on the RQs as shown in Table 17.

Table 17
Comparison of automatic and manual assumptions management.

Aspect	Automatic assumptions management	Manual assumptions management
Definition	The studies related to AAM usually do not define the assumption concept.	There are several explicit definitions of ‘assumption’ in MAM-related studies.
Objective	AAM is usually used for system verification, including modular, component, and program verification. Assumptions are automatically managed in AAM.	MAM is used for various purposes, including managing assumptions in the five software development activities (see Appendix B). MAM usually need the involvement of stakeholders (e.g., making assumptions by architects).
Software development activity	AAM is mainly used in software design and construction and rarely used in other software development activities.	MAM is mainly used in requirements engineering and software design; moderately used in other software development activities.
Software artifact	The most related artifact of AAM is component. Other types of artifacts such as interface, source code, thread, model, and requirement can also be involved.	The artifacts related to MAM are various, including requirement, architecture, design decision, component, model, and source code.
Classification of assumptions	AAM-related studies usually do not classify assumptions.	There are certain classifications of assumptions identified in MAM-related studies.
Assumptions management activity	The main assumptions management activities in AAM are Making, Description, and Evaluation. The other activities are rarely used in AAM.	The main assumptions management activities in MAM are Making, Description, Evaluation, and Maintenance. The other activities can be used in MAM.
Approach	The representative approach in AAM is assume-guarantee reasoning.	There are no representative approaches in MAM.
Tool	There are certain tools dedicated for AAM.	Most of the identified tools for MAM are either general tools in software development, or aim at managing other types of artifacts but can be used for MAM.
Stakeholder	The studies related to AAM usually do not mention who should be involved.	The studies related to MAM include various types of stakeholders, such as customer, requirements engineer, designer, and developer.
Benefit	Most of the benefits mentioned by the AAM-related studies are for software design, construction, and verification.	MAM can benefit all the five software development activities (see Appendix B) as well as other aspects such as improving software quality and traceability between artifacts.
Challenge	Most of the challenges in AAM are related to assumptions management itself, such as how to handle the assumptions explosion problem.	Challenges of MAM are related not only to assumptions management itself, but also the assumption concept, such as understanding and classification of assumptions.
Consequence of not well-managed assumptions	The studies related to AAM usually do not mention the consequences caused by not well-managed assumptions.	The studies related to MAM identified various consequences of not well-managed assumptions, such as system failures and design-related problems.
Lessons learned	Most of the lessons learned in AAM are suggestions for assumptions management, especially suggestions regarding how to use the related approaches, tools, etc.	The lessons learned of MAM cover all the three aspects in Table 14, i.e., impact of assumptions, suggestions for assumptions management, and understanding of assumptions.

5.3. Implications for researchers

- (1) As shown in Fig. 6, 29.9% of the studies have practitioners as authors. According to the results of this SMS, assumptions as well as their management play an important role in software development, and we encourage researchers to collaborate more with practitioners in their studies.
- (2) Based on the presented evidence, there are only a few studies that make the “*assumption*” concept clear, while most of the studies do not provide any definition. Researchers need to explain and disambiguate the concept and the term “*assumption*” in their own work.
- (3) Though we advocated classifying assumptions based on the five software development activities from SWEBOK [12], there is a need to verify its suitability or further adapt this classification in practice.
- (4) Two potential reasons (see Point (2) of Section 5.1) are provided regarding why the majority of assumptions in software development are made or related to the artifacts in requirements engineering and software design. However, there is a need to confirm these reasons in further research.
- (5) Assumptions are related to various types of software artifacts. Nine types of relationships between assumptions and certain types of artifacts were identified. There is a need to study in more detail why, how, and when assumptions can impact certain types of artifacts or vice versa.
- (6) Not all the assumptions management activities received the same attention according to the studies. It is valuable to identify and analyze the benefits, costs, or other factors of the assumptions management activities in software development.
- (7) The approaches and tools for assumptions management have their context of use. Researchers need to make clear the context of use when introducing a new assumptions management approach or tool. Furthermore, no studies employed Assumptions

Table 18
Number of studies per publication venue.

Publication venue	Type	Number (%%)
International Conference on Computer Aided Verification (CAV)	Conference	6 (4.5%%)
Journal of Systems and Software	Journal	5 (3.7%%)
International Requirements Engineering Conference (RE)	Conference	4 (3.0%%)
International Conference on Automated Software Engineering (ASE)	Conference	4 (3.0%%)
Asia-Pacific Software Engineering Conference (APSEC)	Conference	4 (3.0%%)
International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)	Conference	4 (3.0%%)
International Conference on Model Driven Engineering Languages and Systems (MoDELS)	Conference	4 (3.0%%)
Formal Methods in System Design	Journal	3 (2.2%%)
International Conference on Software Engineering (ICSE)	Conference	3 (2.2%%)
International Workshop on Software Evolvability (SE)	Workshop	3 (2.2%%)
IEEE Transactions on Software Engineering	Journal	2 (1.5%%)
IET Software	Journal	2 (1.5%%)
ACM Transactions on Software Engineering and Methodology	Journal	2 (1.5%%)
Science of Computer Programming	Journal	2 (1.5%%)
Requirements Engineering	Journal	2 (1.5%%)
International Conference on Computer Safety, Reliability, and Security (SAFECOMP)	Conference	2 (1.5%%)
Annual International Computer Software and Applications Conference (COMPSAC)	Conference	2 (1.5%%)
International Conference on Pervasive Services (ICPS)	Conference	2 (1.5%%)
International Symposium on Software Reliability Engineering (ISSRE)	Conference	2 (1.5%%)
International Workshop on the Twin Peaks of Requirements and Architecture (TwinPeaks)	Workshop	2 (1.5%%)
Workshop on Sharing and Reusing Architectural Knowledge (SHARK)	Workshop	2 (1.5%%)

Communication, Reuse, Understanding, Searching, and Organization, and most of the tools were used for assume-guarantee reasoning (19 out of 34, 55.9%%) and Assumptions Description (12 out of 15, 80.0%%, 15 is the number of the tools not used for assume-guarantee reasoning, see [Section 4.6.2](#)). There is a need to explore the possibility of employing these assumptions management activities in software development, and provide more dedicated tools, especially in cooperation with industry.

- (8) [Table 9](#) shows that all the types of stakeholders have been involved in Assumptions Making, followed by Evaluation and Description; the stakeholders of requirements engineering, software design, and software construction play a central role in assumptions management. Besides conducting assumptions management activities, we also identified another four types of relationships between stakeholders and assumptions management as presented in [Section 4.7](#). This result indicates that researchers need to pay attention to different stakeholders when integrating assumptions management in software development. Additionally, we argue that the stakeholders and their related assumptions management activities collected from the selected studies are not complete. For example, there are no studies that explicitly mention which stakeholders should be involved in Assumptions Reuse and Searching. One reason may be that “stakeholders” were not a major concern in the studies. Researchers are encouraged to delve into this topic, in order to make clear, for example, the responsibilities of stakeholders in assumptions management.
- (9) The benefits of assumptions management are mostly based on certain conditions (e.g., the approaches or tools used). Researchers need to present the factors that may impact on the benefits when introducing their work.
- (10) Most negative consequences were caused by invalid or implicit assumptions, and we suggest that researchers focus on how to make assumptions explicit and reduce the number of invalid assumptions.
- (11) There is a significant difference between automatic and manual assumptions management, regarding their pros, cons, characteristics, etc. An interesting topic is to combine automatic and manual assumptions management (e.g., using assume-guarantee approaches with different types of stakeholders involved).

5.4. Implications for practitioners

- (1) Practitioners need to reach an agreement on the understanding of assumptions in their own context, organization, or projects, in order

to reduce the ambiguity of the assumption concept and term.

- (2) Practitioners need to understand that assumptions should be managed from the early phases of software development. We suggest that practitioners reflect on when exactly to manage assumptions and which assumptions are more critical to manage in software development.
- (3) Assumptions are related to various types of software artifacts, and there are twelve assumptions management activities identified and collected in this SMS. We suggest that practitioners make explicit decisions on how to manage assumptions based on their available resources. For example, if resources are scarce, it is more beneficial to only include Assumptions Making, Description, Evaluation, and Maintenance, than covering all the assumptions management activities.
- (4) The approaches, tools, benefits, and lessons learned of assumptions management have their own context as detailed in [Section 5.1](#). For example, practitioners need to consider critically the benefits and lessons learned of assumptions management, regarding their specific context.
- (5) The challenges of assumptions management (see [Section 4.8.3](#)) and the consequences of not well-managed assumptions (see [Section 4.9](#)) can act as a checklist for practitioners. We suggest that practitioners become aware of the challenges and the negative consequences when assumptions are not well managed.
- (6) Practitioners make assumptions in their work without being aware of them, and therefore these assumptions stay implicit, and no one knows whether and when they turn to be invalid. Having a fair understanding of assumptions, acknowledging that practitioners make assumptions in their work, and perceiving the existing assumptions are the first significant steps of assumptions management in practice.

6. Threats to validity

The validity threats of this SMS exist during all the phases of conducting the study. This section discusses the threats according to the SMS phases, i.e., study search and selection (see [Section 6.1](#)), data extraction (see [Section 6.2](#)), and data analysis (see [Section 6.3](#)).

6.1. Study search and selection

We may have omitted related studies due to several reasons such as personal bias during the selection phases; this may have a negative

Table 19
Types of assumptions related to software development activities.

Software development activity	Number (%)	Studies
Requirements engineering	76 (56.7%%)	[S2][S3][S4][S5][S6][S7][S8][S9][S13][S16][S17][S18][S29][S30][S32][S34][S36][S38][S40][S43][S44][S45][S46][S48][S50][S53][S54][S56][S59][S60][S61][S62][S63][S64][S66][S67][S70][S73][S74][S76][S78][S80][S82][S84][S85][S88][S89][S90][S91][S92][S93][S96][S98][S99][S100][S102][S103][S104][S105][S106][S107][S108][S109][S111][S113][S114][S115][S116][S119][S121][S122][S125][S126][S127][S128][S133]
Software design	104 (77.6%%)	[S2][S3][S4][S6][S7][S8][S9][S10][S11][S12][S13][S14][S15][S17][S18][S19][S20][S21][S22][S23][S24][S25][S26][S27][S29][S30][S31][S33][S34][S35][S36][S37][S39][S40][S41][S43][S44][S46][S49][S50][S52][S53][S54][S55][S56][S57][S58][S59][S60][S64][S65][S66][S67][S68][S69][S70][S71][S72][S73][S75][S76][S77][S78][S80][S81][S83][S84][S85][S86][S87][S88][S89][S90][S91][S92][S93][S94][S95][S97][S99][S102][S103][S104][S105][S109][S110][S111][S112][S114][S116][S117][S119][S120][S121][S122][S123][S124][S126][S127][S130][S131][S132][S133][S134]
Software construction	36 (26.9%%)	[S1][S2][S3][S6][S9][S13][S17][S23][S34][S36][S40][S42][S44][S47][S50][S51][S54][S55][S57][S75][S76][S80][S86][S87][S92][S93][S100][S101][S105][S109][S111][S114][S119][S123][S124][S133]
Software testing	16 (11.9%%)	[S7][S11][S13][S36][S40][S50][S54][S55][S80][S92][S93][S100][S105][S114][S120][S133]
Software maintenance and evolution	22 (16.4%%)	[S4][S5][S8][S20][S34][S50][S71][S72][S80][S84][S85][S92][S93][S100][S105][S114][S115][S116][S122][S123][S132][S133]

Table 20
Assumptions management activities in software development.

Assumptions management activity	Number (%)	Studies
Making	108 (80.6%%)	[S1][S2][S3][S4][S6][S7][S9][S11][S12][S13][S14][S16][S17][S19][S20][S21][S22][S23][S24][S27][S28][S29][S33][S34][S35][S37][S38][S39][S40][S41][S42][S43][S44][S45][S46][S48][S49][S50][S51][S52][S53][S54][S55][S56][S57][S58][S59][S60][S61][S62][S63][S65][S67][S68][S69][S70][S71][S72][S73][S74][S75][S76][S78][S79][S80][S81][S82][S83][S84][S85][S86][S87][S88][S89][S90][S91][S92][S93][S97][S99][S100][S101][S102][S103][S104][S105][S106][S108][S109][S110][S111][S112][S114][S115][S116][S118][S120][S121][S123][S124][S125][S126][S127][S130][S131][S132][S133][S134]
Description	89 (66.4%%)	[S1][S2][S3][S4][S5][S6][S7][S10][S16][S17][S18][S22][S23][S25][S26][S27][S31][S33][S34][S35][S36][S37][S38][S43][S44][S45][S46][S47][S48][S49][S51][S54][S55][S56][S57][S59][S60][S61][S64][S66][S67][S68][S73][S74][S75][S76][S77][S78][S79][S80][S82][S83][S84][S85][S86][S89][S90][S92][S93][S96][S97][S98][S99][S100][S101][S103][S104][S105][S107][S108][S109][S111][S112][S113][S114][S116][S117][S118][S120][S122][S123][S124][S125][S126][S127][S128][S130][S132][S133]
Evaluation	83 (61.9%%)	[S1][S2][S4][S5][S6][S10][S13][S14][S15][S17][S19][S20][S23][S25][S27][S29][S30][S33][S34][S35][S38][S39][S40][S42][S43][S44][S46][S47][S51][S53][S54][S55][S56][S57][S58][S61][S62][S63][S67][S70][S71][S73][S75][S76][S77][S78][S79][S80][S81][S83][S84][S87][S90][S92][S93][S94][S95][S97][S98][S99][S100][S101][S102][S103][S104][S109][S111][S113][S115][S116][S118][S119][S120][S123][S124][S125][S126][S127][S128][S129][S132][S133][S134]
Maintenance	30 (22.4%%)	[S4][S5][S6][S10][S14][S19][S22][S26][S29][S34][S38][S41][S43][S50][S52][S64][S80][S81][S86][S87][S90][S92][S93][S97][S101][S103][S113][S114][S132][S133]
Tracing	19 (14.2%%)	[S5][S34][S36][S43][S54][S60][S67][S85][S89][S90][S96][S104][S105][S114][S115][S122][S123][S126][S133]
Monitoring	18 (13.4%%)	[S1][S4][S18][S31][S34][S55][S78][S84][S93][S97][S99][S100][S104][S105][S114][S119][S120][S125]
Communication	10 (7.5%%)	[S2][S67][S78][S79][S104][S105][S101][S111][S123][S132]
Reuse	9 (6.7%%)	[S20][S49][S59][S71][S72][S90][S105][S110][S133]
Understanding	8 (6.0%%)	[S6][S63][S89][S90][S96][S121][S124][S132]
Recovery	6 (4.5%%)	[S34][S88][S105][S114][S132][S133]
Searching	5 (3.7%%)	[S92][S93][S105][S119][S132]
Organization	3 (2.2%%)	[S78][S108][S111]

impact on the results of this SMS. The following strategies were used to reduce the risks.

- (1) Seven popular databases (e.g., IEEE Explore) on software engineering were included for the database search and we also used Snowballing for manual search.
- (2) We designed and reached an agreement on inclusion and exclusion criteria (see Section 3.2.2.4) for selecting studies, which helped to avoid misunderstandings between the researchers.
- (3) A trial search and selection was conducted to enhance the quality (e.g., correctness) of this SMS (see Section 3.2.1) before the formal search and selection.
- (4) Two researchers conducted all the selection phases independently in the formal search and selection, and the results were reviewed by another two researchers.
- (5) Furthermore, before conducting each phase in both the trial and formal search and selection, the researchers had a meeting to resolve problems (e.g., misunderstandings and conflicts between researchers) of the previous phase and reached an agreement about the execution of the next phase.

Another threat is that we may have missed primary papers published before the year 2001. As discussed in Section 3.2.2.2, to the best of our knowledge, there are no papers that can potentially act as milestones of assumptions and their management in software development. Furthermore, we considered that 15 years is a reasonable period for the topic of this SMS.

6.2. Data extraction

Personal bias may decrease the quality of the extracted data from the studies (e.g., incompleteness of the extracted data). The strategies used to mitigate this threat are the following:

- (1) We designed and reached an agreement on the data items.
- (2) Potential problems in data extraction (e.g., whether certain data should be extracted) were discussed by the researchers. For example, we might find certain data related to system failure, however, it is not caused by not well-managed assumptions, but other reasons. Then this data should not be extracted.
- (3) A trial data extraction was conducted before the formal data

Table 21
Approaches for assumptions management.

Approach	Number (%)	Studies
Assumptions Making	62 (46.3%%)	[S6][S7][S9][S11][S12][S13][S14][S15][S19][S20][S21][S22][S27][S29][S34][S35][S37][S38][S40][S41][S45][S48][S51][S52][S53][S54][S55][S58][S59][S62][S61][S63][S65][S67][S70][S71][S72][S73][S76][S81][S82][S83][S86][S94][S95][S97][S100][S102][S103][S104][S106][S109][S110][S111][S116][S123][S124][S126][S130][S131][S132][S134]
Assumptions Description	65 (48.5%%)	[S1][S5][S7][S10][S16][S18][S22][S23][S26][S27][S31][S33][S34][S36][S37][S38][S43][S44][S45][S46][S47][S48][S51][S55][S56][S57][S59][S60][S61][S64][S66][S67][S73][S74][S75][S77][S82][S84][S85][S86][S90][S96][S97][S98][S100][S101][S103][S104][S105][S107][S108][S111][S112][S115][S116][S117][S119][S120][S122][S124][S126][S127][S128][S132][S133]
Assumptions Evaluation	40 (29.9%%)	[S1][S6][S10][S13][S15][S17][S20][S27][S29][S30][S34][S47][S51][S53][S54][S55][S56][S57][S58][S63][S70][S71][S73][S76][S77][S83][S94][S95][S97][S101][S102][S103][S109][S111][S116][S123][S124][S126][S129][S134]
Assumptions Maintenance	11 (8.2%%)	[S10][S14][S22][S26][S29][S34][S81][S86][S97][S101][S103]
Assumptions Tracing	5 (3.7%%)	[S36][S60][S96][S122][S126]
Assumptions Monitoring	3 (2.2%%)	[S1][S4][S104]
Assumptions Recovery	1 (0.7%%)	[S114]

extraction.

- (4) The data was extracted by one researcher, and was checked and confirmed by another researcher.

6.3. Data analysis

Data analysis can be potentially impacted by personal bias, mistakes (e.g., because of carelessness), and low quality of the extracted data. The following strategies were used to mitigate the threats:

- (1) Potential problems in data analysis were discussed by the researchers. Two representative examples: whether the data of a RQ should be analyzed through descriptive statistics or Constant Comparison; whether a code of qualitative data is correct and precise.
- (2) The data was analyzed by one researcher, and was checked and confirmed by another researcher.
- (3) The data without a clear description was excluded from the results of this SMS.
- (4) The MAXQDA tool⁴ used in this SMS is suitable especially for qualitative data analysis. For example, using its coding system and visualization tools can reduce the threat of mistakes in data analysis.

7. Conclusions

In this work, we conducted an SMS that covers the publications from January 2001 to December 2015 on the topic of assumptions and their management in software development. 19,428 papers were retrieved from the databases in the first place, and a final set of 134 papers was selected for further data extraction and analysis. This SMS provides a comprehensive overview of the state of the research and practice, acts as a foundation for future work on this topic, and helps practitioners to better understand and manage assumptions in their work.

We draw conclusions in nine aspects:

- (1) The studies were published in 94 venues, which indicates that assumptions and their management has been a broad topic in software engineering. Most of the studies were published in conferences (81 out of 134, 60.4%%), followed by journals (34 out of 134, 25.4%%). However, the top two venues in Table 18 published only six (out of 134, 4.5%%) and five (out of 134, 3.7%%) studies, respectively, and there are no venues that specifically publish studies on assumptions and their management in software development.

- (2) Only twenty-one studies (out of 134, 15.7%%) explicitly defined the assumption concept. Though the assumption concept may be expected as common knowledge in software development, it is rather important to make this concept clear in both research and practice.
- (3) Most assumptions are made or related to the artifacts in requirements engineering and software design, which demonstrates that assumptions should be managed from the early phases of software development. However, the rationale behind this phenomenon is unclear, and therefore, more investigation is needed. Furthermore, eight types of relationships between assumptions and other types of artifacts were identified in this SMS. There is a need to maintain these relationships as well as to identify new relationships in further studies.
- (4) Though twelve assumptions management activities were explored in the selected studies of this SMS, much effort (e.g., the related approaches) has been put on Assumptions Making, Description, and Evaluation. Assumptions Maintenance received moderate attention. As discussed in Section 5.1, these four activities could be considered as primary activities in assumptions management. However, this conclusion needs further validation.
- (5) More than half of the tools identified from the studies are aimed at supporting assume-guarantee reasoning. For the other tools, most of them can be used to support Assumptions Description. As assume-guarantee reasoning is a specific part within the topic of assumptions and their management in software development, there is a clear need to develop more dedicated tools, especially tools that support not only Description, but also other assumptions management activities, such as Tracing.
- (6) All the identified types of stakeholders are involved in Assumptions Making in their work, followed by Evaluation and Description. Stakeholders involved in requirements engineering, software design, and software construction play a central role in assumptions management. On the contrary, there are no studies that explicitly mention which stakeholders should be involved in Assumptions Reuse, Searching, and Organization. There is a need to further explore the rationale behind this phenomenon, as well as to develop a clear map between stakeholders and assumptions management activities.
- (7) As identified in this SMS, the main challenge is the difficulty of performing assumptions management activities in software development. Addressing this challenge should be considered as the most burning issue when managing assumptions.
- (8) The identified assumptions management approaches, tools, benefits, and lessons learned are limited to their specific contexts (e.g., context of use or preconditions). As an example, though certain benefits of managing assumptions in software development were identified from the selected studies, they are not for free, but depend on various conditions (e.g., using specific assumptions

⁴ <http://www.maxqda.com/>.

management approaches, activities, or tools), or may raise certain challenges in assumptions management (e.g., how to balance the benefits of assumptions management and the required effort). This finding motivates further investigation on generalizability of such approaches, tools, benefits, and lessons learned, and return on investment of assumptions management in software development.

- (9) Most of the negative consequences identified in this SMS are caused by invalid or implicit assumptions. Therefore, one conclusion is that reducing invalid and implicit assumptions is critical in software development.

Besides the detailed discussion on future work in Section 5, we plan to conduct more studies on assumptions and their management in software development, including interplay between assumptions and other types of software artifacts, assumptions management process,

approaches and tools dedicated to supporting various assumptions management activities, involvement of various stakeholders in assumptions management, solutions for assumptions management challenges, and transforming the lessons learned of assumptions management into practices.

Acknowledgments

This work is partially sponsored by the NSFC under Grant No. 61472286 and the Ubbo Emmius scholarship program by the University of Groningen. The authors wish to thank Zhuang Xiong, Tianqing Liu, Julien van der Land, and Stephan Boomker for their study search and selection effort in this work. We would also like to thank Christian Manteuffel for reviewing an earlier version of the paper.

Appendix A. Selected studies

- [S1] A. Askarov, S. Chong, and H. Mantel. Hybrid monitors for concurrent noninterference. In: *Proceedings of the 28th Computer Security Foundations Symposium (CSF)*, Verona, Italy, pp. 137–151, 2015.
- [S2] Ö. Albayrak, D. Albayrak, and T. Kılıç. Are software engineers' responses to incomplete requirements related to project characteristics? In: *Proceedings of the 2nd International Conference on the Applications of Digital Information and Web Technologies (ICADIWT)*, London, UK, pp. 124–129, 2009.
- [S3] Ö. Albayrak, H. Kurtoğlu, and M. Bıçakçı. Incomplete software requirements and assumptions made by software engineers. In: *Proceedings of the 16th Asia-Pacific Software Engineering Conference (APSEC)*, Penang, Malaysia, pp. 333–339, 2009.
- [S4] R. Ali, F. Dalpiaz, P. Giorgini, and V.E.S. Souza. Requirements evolution: from assumptions to reality. In: *Proceedings of the 12th International Conference on Enterprise, Business-Process and Information Systems Modeling (BPMDS)*, London, UK, pp. 372–382, 2011.
- [S5] J. Arnowitz, M. Arent, and N. Berger. Verify Prototype Assumptions and Requirements. In: *User Experience Re-Mastered: Your Guide to Getting the Right Design*, Elsevier, pp. 221–240, 2010.
- [S6] J.D. Arthur, R.E. Nance, A. Bazaz, and O. Balci. Mitigating security risks in systems that support pervasive services and computing: Access-driven verification, validation and testing. In: *Proceedings of the 22th IEEE International Conference on Pervasive Services (ICPS)*, Istanbul, Turkey, pp. 109–117, 2007.
- [S7] T. Arts, M. Dorigatti, and S. Tonetta. Making implicit safety requirements explicit. In: *Proceedings of the 33rd International Conference on Computer Safety, Reliability, and Security (SAFECOMP)*, Florence, Italy, pp. 81–92, 2014.
- [S8] M. Autili, P. Inverardi, and M. Tivoli. Assessing Dependability for Mobile and Ubiquitous Systems: Is There a Role for Software Architectures? In: *Patterns, Programming and Everything*. Springer London, pp. 1–12, 2010.
- [S9] I. Bate and N. Audsley. Flexible design of complex high-integrity systems using trade offs. In: *Proceedings of the 8th IEEE International Symposium on High Assurance Systems Engineering (HASE)*, Tampa, FL, USA, pp. 22–31, 2004.
- [S10] S. Bauer, R. Hennicker, and A. Legay. A meta-theory for component interfaces with contracts on ports. *Science of Computer Programming*, 91(10): 70–89, 2014.
- [S11] A. Bazaz, J.D. Arthur, and J.G. Tront. Modeling security vulnerabilities: A constraints and assumptions perspective. In: *Proceedings of the 2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing (DASC)*, Indianapolis, IN, USA, pp. 95–102, 2006.
- [S12] J. Bhuta and B. Boehm. A framework for identification and resolution of interoperability mismatches in cots-based systems. In: *Proceedings of the 2nd International Workshop on Incorporating COTS Software into Software Systems: Tools and Techniques (IWICSS)*, Minneapolis, MN, USA, Article No. 2, 2007.
- [S13] C. Blundell, D. Giannakopoulou, and C.S. Păsăreanu. Assume-guarantee testing. *ACM SIGSOFT Software Engineering Notes*, 31(2): Article No. 1, 2005.
- [S14] M.G. Bobaru, C.S. Păsăreanu, and D. Giannakopoulou. Automated assume-guarantee reasoning by abstraction refinement. In: *Proceedings of the 20th International Conference on Computer Aided Verification (CAV)*, Princeton, NJ, USA, pp. 135–148, 2008.
- [S15] S. Bogomolov, G. Frehse, M. Greitschus, R. Grosu, C. Pasareanu, A. Podelski, and T. Strump. Assume-guarantee abstraction refinement meets hybrid systems. In: *Proceedings of the 10th International Haifa Verification Conference on Hardware and Software: Verification and Testing (HVC)*, Haifa, Israel, pp. 116–131, 2014.
- [S16] C. Brenner, J. Greenyer, and V. Panzica La Manna. The ScenarioTools play-out of modal sequence diagram specifications with environment assumptions. *Electronic Communications of the EASST*, 58, 2013.
- [S17] J.Y. Brunel, M. Di Natale, A. Ferrari, P. Giusto, and L. Lavagno. Softcontract: an assertion-based software development process that enables design-by-contract. In: *Proceedings of the 11th Conference on Design, Automation and Test in Europe (DATE)*, Paris, France, pp. 358–363, 2004.
- [S18] D. Bush and A. Finkelstein. Requirements stability assessment using scenarios. In: *Proceedings of the 11th IEEE International Requirements Engineering Conference (RE)*, Monterey Bay, CA, USA, pp. 23–32, 2003.
- [S19] G. Bobeff and J. Noyé. Component specialization. In: *Proceedings of the 8th ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation (PEPM)*, Verona, Italy, pp. 39–50, 2004.
- [S20] S. Chaki, E. Clarke, N. Sharygina, and N. Sinha. Verification of evolving software via component substitutability analysis. *Formal Methods in System Design*, 32(3): 235–266, 2008.
- [S21] S. Chaki, E. Clarke, N. Sinha, and P. Thati. Automated assume-guarantee reasoning for simulation conformance. In: *Proceedings of the 17th International Conference on Computer Aided Verification (CAV)*, Edinburgh, Scotland, UK, pp. 534–547, 2005.
- [S22] A. Chakrabarti, L. de Alfaro, T.A. Henzinger, and F.Y.C. Mang. Synchronous and bidirectional component interfaces. In: *Proceedings of the 14th International Conference on Computer Aided Verification (CAV)*, Copenhagen, Denmark, pp. 414–427, 2002.

- [S23] A. Chakrabarti, L. de Alfaro, T.A. Henzinger, M. Jurdziński, and F.Y.C. Mang. Interface compatibility checking for software modules. In: Proceedings of the 14th International Conference on Computer Aided Verification (CAV), Copenhagen, Denmark, pp. 428–441, 2002.
- [S24] G. Chroust. The empty chair: uncertain futures and systemic dichotomies. *Systems Research and Behavioral Science*, 21(3): 227–236, 2004.
- [S25] A. Cimatti, M. Dorigatti, and S. Tonetta. OCRA: A tool for checking the refinement of temporal contracts. In: Proceedings of the 28th IEEE/ACM 28th International Conference on Automated Software Engineering (ASE), Silicon Valley, CA, USA, pp. 702–705, 2013.
- [S26] A. Cimatti and S. Tonetta. A property-based proof system for contract-based design. In: Proceedings of the 38th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Cesme, Izmir, Turkey, pp. 21–28, 2012.
- [S27] A. Cimatti and S. Tonetta. Contracts-refinement proof system for component-based embedded systems. *Science of Computer Programming*, 97(1): 333–348, 2015.
- [S28] J.M. Cobleigh, G.S. Avrunin, and L.A. Clarke. Breaking up is hard to do: An evaluation of automated assume-guarantee reasoning. *ACM Transactions on Software Engineering and Methodology*, 17(2): Article No. 7, 2007.
- [S29] J.M. Cobleigh, D. Giannakopoulou, and C.S. Păsăreanu. Learning assumptions for compositional verification. In: Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS) Held as Part of the Joint European Conferences on Theory and Practice of Software (ETAPS), Warsaw, Poland, pp. 331–346, 2003.
- [S30] D. Cofer, A. Gacek, S. Miller, M.W. Whalen, B. LaValley, and L. Sha. Compositional verification of architectural models. In: Proceedings of the 4th International Symposium on NASA Formal Methods (NFM), Norfolk, VA, USA, pp. 126–140, 2012.
- [S31] S. Cohen, W. Money, and S. Kaisler. Service migration in an enterprise system architecture. In: Proceedings of the 42nd Hawaii International Conference on System Sciences (HICSS), Big Island, HI, USA, pp. 1–10, 2009.
- [S32] M. Daneva and R. Wieringa. Requirements engineering for cross-organizational ERP implementation undocumented assumptions and potential mismatches. In: Proceedings of the 13th IEEE International Conference on Requirements Engineering (RE), Paris, France, pp. 63–72, 2005.
- [S33] L. de Alfaro and M. Stoelinga. Interfaces: A game-theoretic framework for reasoning about component-based systems. *Electronic Notes in Theoretical Computer Science*, 97(7): 3–23, 2004.
- [S34] V. De Florio. Software Assumptions Failure Tolerance: Role, Strategies, and Visions. In: *Architecting Dependable Systems VII*. Springer Berlin Heidelberg, pp. 249–272, 2010.
- [S35] C. de la Riva and J. Tuya. Automatic generation of assumptions for modular verification of software specifications. *Journal of Systems and Software*, 79(9): 1324–1340, 2005.
- [S36] E. Denney and B. Fischer. A verification-driven approach to traceability and documentation for auto-generated mathematical software. In: Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE), Auckland, New Zealand, pp. 560–564, 2009.
- [S37] P. Derler, E.A. Lee, S. Tripakis, and M. Törngren. Cyber-physical system design contracts. In: Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems (ICCPS), Philadelphia, USA, pp. 109–118, 2013.
- [S38] N. D'ippolito, V. Braberman, N. Piterman, and S. Uchitel. Synthesizing nonanomalous event-based controllers for liveness goals. *ACM Transactions on Software Engineering and Methodology*, 22(1): Article No. 9, 2013.
- [S39] Z. Dwaikat and F. Parisi-Presicce. Risky trust: risk-based analysis of software systems. In: Proceedings of the 1st Workshop on Software Engineering for Secure Systems (SESS), St. Louis, Missouri, USA, pp. 1–7, 2005.
- [S40] U. Eliasson, R. Heldal, J. Lantz, and C. Berger. Agile model-driven engineering in mechatronic systems - An industrial case study. In: Proceedings of the 17th International Conference on Model-Driven Engineering Languages and Systems (MODELS), Valencia, Spain, pp. 433–449, 2014.
- [S41] M. Emmi, D. Giannakopoulou, and C.S. Păsăreanu. Assume-guarantee verification for interface automata. In: Proceedings of the 15th International Symposium on Formal Methods (FM), Turku, Finland, pp. 116–131, 2008.
- [S42] J. Fabry, C. De Roover, and V. Jonckers. Aspectual source code analysis with GASR. In: Proceedings of the 13th International Working Conference on Source Code Analysis and Manipulation (SCAM), Eindhoven, The Netherlands, pp. 53–62, 2013.
- [S43] S. Faily and I. Fléchaïs. The secret lives of assumptions: Developing and refining assumption personas for secure system design. In: Proceedings of the 3rd International Conference on Human-Centred Software Engineering (HCSE), Reykjavik, Iceland, pp. 111–118, 2010.
- [S44] M. Feilkas and D. Ratiu. Ensuring well-behaved usage of APIs through syntactic constraints. In: Proceedings of the 16th IEEE International Conference on Program Comprehension (ICPC), Amsterdam, The Netherlands, pp. 248–253, 2008.
- [S45] Q. Feng and R. Lutz. Assessing the effect of software failures on trust assumptions. In: Proceedings of the 19th International Symposium on Software Reliability Engineering (ISSRE), Seattle, Redmond, WA, USA, pp. 291–292, 2008.
- [S46] A. Filieri and C. Ghezzi. Further steps towards efficient runtime verification: Handling probabilistic cost models. In: Proceedings of the 1st International Workshop on Formal Methods in Software Engineering: Rigorous and Agile Approaches (FormSERA), Zurich, Switzerland, pp. 2–8, 2012.
- [S47] C. Flanagan, S.N. Freund, and S. Qadeer. Thread-modular verification for shared-memory programs. In: Proceedings of the 11th European Symposium on Programming (ESOP) Held as Part of the Joint European Conferences on Theory and Practice of Software (ETAPS), Grenoble, France, pp. 262–277, 2002.
- [S48] C.H. Fleming and N. Leveson. Integrating systems safety into systems engineering during concept development. *International Council on Systems Engineering Symposium (INCOSE)*, 25(1): 989–1003, 2015.
- [S49] D. Garlan, R. Allen, and J.M. Ockerbloom. Architectural mismatch: Why reuse is still so hard. *IEEE Software*, 26(4): 66–69, 2009.
- [S50] A.Q. Gates and O. Mondragon. FasTLInC: a constraint-based tracing approach. *Journal of Systems and Software*, 63(3): 241–258, 2002.
- [S51] A. Ghabi and A. Egyed. Exploiting traceability uncertainty among artifacts and code. *Journal of Systems and Software*, 108(10): 178–192, 2015.
- [S52] M. Gheorghiu, D. Giannakopoulou, and C.S. Păsăreanu. Refining interface alphabets for compositional verification. In: Proceedings of the 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Braga, Portugal, pp. 292–307, 2007.
- [S53] D. Giannakopoulou, C.S. Păsăreanu, and H. Barringer. Component verification with automatically generated assumptions. *Automated Software Engineering*, 12(3): 297–320, 2005.
- [S54] D. Giannakopoulou, C.S. Păsăreanu, and C. Blundell. Assume-guarantee testing for software components. *IET Software*, 2(6): 547–562, 2008.

- [S55] D. Giannakopoulou, C.S. Păsăreanu, and J.M. Cobleigh. Assume-guarantee verification of source code with design-level assumptions. In: *Proceedings of the 26th International Conference on Software Engineering (ICSE)*, Edinburgh, Scotland, UK, pp. 211–220, 2004.
- [S56] J. Greenyer and E. Kindler. Compositional synthesis of controllers from scenario-based assume-guarantee specifications. In: *Proceedings of the 16th International Conference on Model-Driven Engineering Languages and Systems (MODELS)*, Miami, FL, USA, pp. 774–789, 2013.
- [S57] M. Goldman and S. Katz. MAVEN: Modular aspect verification. In: *Proceedings of the 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, Held as Part of the Joint European Conferences on Theory and Practice of Software (ETAPS), Braga, Portugal, pp. 308–322, 2007.
- [S58] A. Gupta, K.L. McMillan, and Z. Fu. Automated assumption generation for compositional verification. In: *Proceedings of the 19th International Conference on Computer Aided Verification (CAV)*, Berlin, Germany, pp. 420–432, 2007.
- [S59] I. Habli and T. Kelly. Capturing and replaying architectural knowledge through derivational analogy. In: *Proceedings of the 2nd Workshop on SHaring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent (SHARK-ADI)*, Minneapolis, MN, USA: Article No. 4, 2007.
- [S60] C.B. Haley, R.C. Laney, J.D. Moffett, and B. Nuseibeh. Using trust assumptions with security requirements. *Requirements Engineering*, 11(2): 138–151, 2006.
- [S61] C.B. Haley, R.C. Laney, J.D. Moffett, and B. Nuseibeh. Security requirements engineering: A framework for representation and analysis. *IEEE Transactions on Software Engineering*, 34(1): 133–153, 2008.
- [S62] C.B. Haley, J.D. Moffett, R.C. Laney, and B. Nuseibeh. Arguing security: Validating security requirements using structured argumentation. In: *Proceedings of the 3rd Symposium on Requirements Engineering for Information Security (SREIS)* held in conjunction with the 13th International Requirements Engineering Conference (RE), Paris, France, pp. 21–28, 2005.
- [S63] C.B. Haley and B. Nuseibeh. Bridging requirements and architecture for systems of systems. In: *Proceedings of the International Symposium on Information Technology (ITSim)*, Kuala Lumpur, Malaysia, pp. 1–8, 2008.
- [S64] U. Hannemann and J. Hooman. Formal design of real-time components on a shared data space architecture. In: *Proceedings of the 25th IEEE Annual Computer Software and Applications Conference (COMPSAC)*, Chicago, IL, USA, pp. 143–150, 2001.
- [S65] T.A. Henzinger, M. Minea, and V. Prabhu. Assume-guarantee reasoning for hierarchical hybrid systems. In: *Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control (HSCC)*, Rome, Italy, pp. 275–290, 2001.
- [S66] T.M. Hesse and B. Paech. Supporting the collaborative development of requirements and architecture documentation. In: *Proceedings of the 3rd International Workshop on the Twin Peaks of Requirements and Architecture (TwinPeaks)*, Rio de Janeiro, Brazil, pp. 22–26, 2013.
- [S67] T. Heyman, R. Scandariato, and W. Joosen. Security in context: analysis and refinement of software architectures. In: *Proceedings of the 34th IEEE Annual Computer Software and Applications Conference (COMPSAC)*, Seoul, South Korea, pp. 161–170, 2010.
- [S68] R. High, Jr., G. Krishnan, and M. Sanchez. Creating and maintaining coherency in loosely coupled systems. *IBM Systems Journal*, 47(3): 357–376, 2008.
- [S69] K. Hiraishi and P. Kucera. Application of DES theory to verification of software components. *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, E92-A (No. 2): 604–610, 2009.
- [S70] P.A. Hsiung and S.Y. Cheng. Automating formal modular verification of asynchronous real-time embedded systems. In: *Proceedings of the 16th International Conference on VLSI Design (VLSI Design)*, New Delhi, India, pp. 249–254, 2003.
- [S71] P.N. Hung, T. Aoki, and T. Katayama. An effective framework for assume-guarantee verification of evolving component-based software. In: *Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution (IWPSE)*, Amsterdam, The Netherlands, pp. 109–118, 2009.
- [S72] P.N. Hung, V.H. Nguyen, T. Aoki, and T. Katayama. An improvement of minimized assumption generation method for component-based software verification. In: *Proceedings of the 9th IEEE RIVF International Conference on Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF)*, Ho Chi Minh City, Vietnam, pp. 1–6, 2012.
- [S73] P. Inverardi and S. Uchitel. Proving deadlock freedom in component-based programming. In: *Proceedings of the 4th International Conference on Fundamental Approaches to Software Engineering (FASE)*, Genova, Italy, pp. 60–75, 2001.
- [S74] F. Ishikawa, B. Suleiman, K. Yamamoto, and S. Honiden. Physical interaction in pervasive computing: formal modeling, analysis and verification. In: *Proceedings of the 24th International Conference on Pervasive Services (ICPS)*, Split, Croatia, pp. 133–140, 2009.
- [S75] H. Jin and P. Santhanam. An approach to higher reliability using software components. In: *Proceedings of the 12th International Symposium on Software Reliability Engineering (ISSRE)*, Hong Kong, China, pp. 2–11, 2001.
- [S76] J. Kanig, R. Chapman, C. Comar, J. Guitton, Y. Moy, and E. Rees. Explicit assumptions-a prenup for marrying static and dynamic program verification. In: *Proceedings of the 8th International Conference on Tests and Proofs (TAP)*, York, UK, pp. 142–157, 2014.
- [S77] E. Katz and S. Katz. User queries for specification refinement treating shared aspect join points. In: *Proceedings of the 8th IEEE International Conference on Software Engineering and Formal Methods (SEFM)*, Pisa, Italy, pp. 73–82, 2010.
- [S78] D. Klappholz and D. Port. Introduction to MBASE (Model-Based (System) Architecting and Software Engineering). *Advances in Computers*, 62: 203–248, 2004.
- [S79] R. Klendauer, M. Berkovich, R. Gelvin, J.M. Leimeister, and H. Krcmar. Towards a competency model for requirements analysts. *Information Systems Journal*, 22(6): 475–503, 2012.
- [S80] A.J. Ko, M.J. Lee, V. Ferrari, S. Ip, and C. Tran. A case study of post-deployment user feedback triage. In: *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, Waikiki, Honolulu, Hawaii, USA, pp. 1–8, 2011.
- [S81] A. Komuravelli, C.S. Păsăreanu, and E.M. Clarke. Assume-guarantee abstraction refinement for probabilistic systems. In: *Proceedings of the 24th International Conference on Computer Aided Verification (CAV)*, Berkeley, CA, USA, pp. 310–326, 2012.
- [S82] R. Kumar and B.H. Krogh. Heterogeneous verification of embedded control systems. In: *Proceedings of the 47th American Control Conference (ACC)*, Minneapolis, MN, USA, pp. 4597–4602, 2006.
- [S83] M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Assume-guarantee verification for probabilistic systems. In: *Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, Paphos, Cyprus, pp. 23–37, 2010.
- [S84] P. Lago and H. van Vliet. Observations from the recovery of a software product family. In: *Proceedings of the 3rd International Conference on Software Product Lines (SPLC)*, Boston, MA, USA, pp. 214–227, 2004.
- [S85] P. Lago and H. van Vliet. Explicit assumptions enrich architectural models. In: *Proceedings of the 27th International Conference on*

Software Engineering (ICSE), St Louis, Missouri, USA, pp. 206–214, 2005.

[S86] C. Landauer. Wrapping architectures for long-term sustainability. In: Proceedings of the 2nd IEEE International Workshop on Software Evolvability (SE), Philadelphia, PA, USA, pp. 44–49, 2006.

[S87] C. Landauer and K.L. Bellman. Self managed adaptability with wrappings. In: Proceedings of the 1st IEEE International Workshop on Software Evolvability (SE), Budapest, Hungary, pp. 29–34, 2005.

[S88] D.V. Landuyt, E. Truyen, and W. Joosen. On the modularity impact of architectural assumptions. In: Proceedings of the 2012 Workshop on Next Generation Modularity Approaches for Requirements and Architecture (NEMARA), Potsdam, Germany, pp. 13–16, 2012.

[S89] D.V. Landuyt, E. Truyen, and W. Joosen. Documenting early architectural assumptions in scenario-based requirements. In: Proceedings of the Joint 10th Working IEEE/IFIP Conference on Software Architecture and 6th European Conference on Software Architecture (WICSA/ECSA), Helsinki, Finland, pp. 329–333, 2012.

[S90] D.V. Landuyt and W. Joosen. Modularizing early architectural assumptions in scenario-based requirements. In: Proceedings of the 17th International Conference on Fundamental Approaches to Software Engineering (FASE) Grenoble, France, pp. 170–184, 2014.

[S91] D.V. Landuyt and W. Joosen. On the role of early architectural assumptions in quality attribute scenarios: a qualitative and quantitative study. In: Proceedings of the 5th International Workshop on the Twin Peaks of Requirements and Architecture (TwinPeaks), Florence, Italy, pp. 9–15, 2015.

[S92] M.M. Lehman. The role and impact of assumptions in software development, maintenance and evolution. In: Proceedings of the 1st IEEE International Workshop on Software Evolvability (IWSE), Budapest, Hungary, pp. 3–14, 2005.

[S93] M.M. Lehman and J.F. Ramil. Rules and tools for software evolution planning and management. *Annals of Software Engineering*, 11(1): 15–44, 2001.

[S94] J. Li, X. Sun, F. Xie, and X. Song. Component-based abstraction and refinement. In: Proceedings of the 10th International Conference on Software Reuse (ICSR), Beijing, China, pp. 39–51, 2008.

[S95] S.W. Lin, É. André, Y. Liu, J. Sun, and J.S. Dong. Learning assumptions for compositional verification of timed systems. *IEEE Transactions on Software Engineering*, 40(2): 137–153, 2003.

[S96] C. Liu, W. Zhang, H. Zhao, and Z. Jin. Analyzing early requirements of cyber-physical systems through structure and goal modeling. In: Proceedings of the 20th Asia-Pacific Software Engineering Conference (APSEC), Bangkok, Thailand, pp. 140–147, 2013.

[S97] Z. Lu, S. Li, A. Ghose, and P. Hyland. Extending semantic web service description by service assumption. In: Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI), Hong Kong, China, pp. 637–643, 2006.

[S98] S. Maoz and Y. Sa'ar. Assume-guarantee scenarios: semantics and synthesis. In: Proceedings of the 15th International Conference on Model Driven Engineering Languages and Systems (MODELS), Innsbruck, Austria, pp. 335–351, 2012.

[S99] J. Marinčić, A. Mader, and R. Wieringa. Classifying assumptions made during requirements verification of embedded systems. In: Proceedings of the 14th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ), Montpellier, France, pp. 141–146, 2008.

[S100] A. Miranskyy, N. Madhavji, M. Davison, and M. Reesor. Modelling assumptions and requirements in the context of project risk. In: Proceedings of the 13th IEEE International Conference on Requirements Engineering (RE), Paris, France, pp. 471–472, 2005.

[S101] J.P. Near, A. Milicevic, E. Kang, and D. Jackson. A lightweight code analysis and its role in evaluation of a dependability case. In: Proceedings of the 33rd International Conference on Software Engineering (ICSE), Honolulu, HI, USA, pp. 31–40, 2011.

[S102] W. Nam and R. Alur. Learning-based symbolic assume-guarantee reasoning with automatic decomposition. In: Proceedings of the 4th International Symposium on Automated Technology for Verification and Analysis (ATVA), Beijing, China, pp. 170–185, 2006.

[S103] W. Nam, P. Madhusudan, and R. Alur. Automatic symbolic compositional verification by learning assumptions. *Formal Methods in System Design*, 32(3): 207–234, 2008.

[S104] A. Nhlabatsi, Y. Yu, A. Zisman, T. Tun, N. Khan, and A. Bandara. Managing security control assumptions using causal traceability. In: Proceedings of the 8th International Symposium on Software and Systems Traceability (SST), Florence, Italy, pp. 43–49, 2015.

[S105] I. Ostacchini and M. Wermelinger. Managing assumptions during agile development. In: Proceedings of the 4th Workshop on SHaring and Reusing architectural Knowledge (SHARK), Vancouver, BC, Canada, pp. 9–16, 2009.

[S106] V. Page, M. Dixon, and P. Biolkowicz. Object-oriented graceful evolution monitors. In: Proceedings of the 9th International Conference on Object-Oriented Information Systems (OOIS), Geneva, Switzerland, pp. 46–59, 2003.

[S107] V. Page, M. Dixon, and I. Choudhury. Mitigating data gathering obstacles within an agile information systems development environment. In: Proceedings of the 10th International Conference on Intelligent Engineering Systems (INES), London, UK, pp. 11–16, 2006.

[S108] V. Page, M. Dixon, and I. Choudhury. Security risk mitigation for information systems. *BT Technology Journal*, 25(1): 118–127, 2007.

[S109] P. Parizek and F. Plasil. Assume-guarantee verification of software components in sofa 2 framework. *IET Software*, 4(3): 210–211, 2010.

[S110] C.S. Păsăreanu, D. Giannakopoulou, M.G. Bobaru, J.M. Cobleigh, and H. Barringer. Learning to divide and conquer: applying the L* algorithm to automate assume-guarantee reasoning. *Formal Methods in System Design*, 32(3): 175–205, 2008.

[S111] A. Rae, D. Jackson, P. Ramanan, J. Flanz, and D. Leyman. Critical feature analysis of a radiotherapy machine. In: Proceedings of the 22nd International Conference on Computer Safety, Reliability, and Security (SAFECOMP), Edinburgh, UK, pp. 221–234, 2005.

[S112] A.J. Ramirez, B.H.C. Cheng, N. Bencomo, and P. Sawyer. Relaxing claims: Coping with uncertainty while evaluating assumptions at run time. In: Proceedings of the 15th International Conference on Model Driven Engineering Languages and Systems (MODELS), Innsbruck, Austria, pp. 53–69, 2012.

[S113] T. Roehm, B. Bruegge, T.M. Hesse, and B. Paech. Towards identification of software improvements and specification updates by comparing monitored and specified end-user behavior. In: Proceedings of the 29th IEEE International Conference on Software Maintenance (ICSM), Eindhoven, The Netherlands, pp. 464–467, 2013.

[S114] R. Roeller, P. Lago, and H. van Vliet. Recovering architectural assumptions. *Journal of Systems and Software*, 79(4): 552–573, 2006.

[S115] J. Savolainen and J. Kuusela. Consistency management of product line requirements. In: Proceedings of the 5th IEEE International Symposium on Requirements Engineering (RE), Toronto, Ont, Canada, 2001.

[S116] R. Seater, D. Jackson, and R. Gheyi. Requirement progression in problem frames: Deriving specifications from requirements. *Requirements Engineering*, 12(2): 77–102, 2007.

[S117] R. Shukla, D. Carrington, and P. Strooper. Systematic operational profile development for software components. In: Proceedings of the

11th Asia-Pacific Software Engineering Conference (APSEC), Busan, South Korea, pp. 528–537, 2004.

[S118] T. Srivatanakul, J.A. Clark, S. Stepney, and F. Polack. Challenging formal specifications by mutation: a CSP security example. In: Proceedings of the 10th Asia-Pacific Software Engineering Conference (APSEC), Bangkok, Thailand, pp. 340–350, 2003.

[S119] A. Steingruebl and G. Peterson. Software assumptions lead to preventable errors. *IEEE Security & Privacy*, 7(4): 84–87, 2009.

[S120] J. Sun, Y. Liu, J.S. Dong, and H.H. Wang. Specifying and verifying event-based fairness enhanced systems. In: Proceedings of the 10th International Conference on Formal Engineering Methods (ICFEM), Kitakyushu-City, Japan, pp. 5–24, 2008.

[S121] M. Svahnberg, T. Gorschek, M. Eriksson, A. Borg, K. Sandahl, J. Borster, and A. Loconsole. Perspectives on requirements understandability - For whom does the teacher's bell toll? In: Proceedings of the 3rd International Workshop on Requirements Engineering Education and Training (REET), Barcelona, Spain, pp. 22–29, 2008.

[S122] A. Tang, Y. Jin, and J. Han. A rationale-based architecture model for design traceability and reasoning. *Journal of Systems and Software*, 80(6): 918–934, 2007.

[S123] A. Tirumala, T. Crenshaw, L. Sha, G. Baliga, S. Kowshik, C. Robinson, and W. Witthawaskul. Prevention of failures due to assumptions made by software components in real-time systems. *ACM SIGBED Review - Special Issue: The 2nd Workshop on High Performance, Fault Adaptive, Large Scale Embedded Real-Time Systems (FALSE-II)*, 2(3): 36–39, 2005.

[S124] O. Tkachuk, M.B. Dwyer, and C.S. Păsăreanu. Automated environment generation for software model checking. In: Proceedings of the 18th IEEE/ACM International Conference on Automated Software Engineering (ASE), Montreal, Canada, pp. 116–127, 2003.

[S125] T.T. Tun, R. Laney, Y. Yu, and B. Nuseibeh. Specifying software features for composition: A tool-supported approach. *Computer Networks*, 57(12): 2454–2464, 2013.

[S126] F. Warg, B. Vedder, M. Skoglund, and A. Soderberg. SafetyADD: A tool for safety-contract based design. In: Proceedings of the 25th IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Naples, Italy, pp. 527–529, 2014.

[S127] K. Welsh, P. Sawyer, and N. Bencomo. Towards requirements aware systems: Run-time resolution of design-time assumptions. In: Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE), Lawrence, KS, USA, pp. 560–563, 2011.

[S128] S. Winetzhammer, J. Greenyer, and M. Tichy. Integrating graph transformations and modal sequence diagrams for specifying structurally dynamic reactive systems. In: Proceedings of the 8th International Conference on System Analysis and Modeling: Models and Reusability (SAM), Valencia, Spain, pp. 126–141, 2014.

[S129] H. Winschiers and J. Fendler. Assumptions considered harmful. In: Proceedings of the 2nd International Conference on Usability and Internationalization (UI-HCII), Beijing, China, pp. 452–461, 2007.

[S130] F. Xie and J.C. Browne. Verification of component-based software application families. In: Proceedings of the 9th International Symposium on Component-Based Software Engineering (CBSE), Västerås, Sweden, pp. 50–66, 2006.

[S131] Q. Yang, E.M. Clarke, A. Komuravelli, and M. Li. Assumption generation for asynchronous systems by abstraction refinement. In: Proceedings of the 9th International Symposium on Formal Aspects of Component Software (FACS), Mountain View, CA, USA, pp. 260–276, 2012.

[S132] C. Yang and P. Liang. Identifying and recording software architectural assumptions in agile development. In: Proceedings of the 26th International Conference on Software Engineering and Knowledge Engineering (SEKE), Vancouver, Canada, pp. 308–313, 2014.

[S133] S. Zschaler and A. Rashid. Aspect assumptions: A retrospective study of AspectJ developers' assumptions about aspect usage. In: Proceedings of the 10th International Conference on Aspect-Oriented Software Development (AOSD), Porto de Galinhas, Pernambuco, Brazil, pp. 93–104, 2011.

[S134] M. Zulkernine and R.E. Seviara. Assume-guarantee supervisor for concurrent systems. In: Proceedings of the 15th International Parallel & Distributed Processing Symposium (IPDPS), San Francisco, USA, pp. 151–159, 2001.

Appendix B. Software development activities

Software development is composed of several basic activities [12]. According to our knowledge and experience, we adapted and followed the software development activities suggested in SWEBOK [12] in this SMS as follows.

(1) Requirements engineering

Software requirements refer to the concerns of various stakeholders related to software products. Requirements engineering includes several sub-activities, such as requirements elicitation, requirements analysis, and requirements specification.

(2) Software design

Software design can be further classified into architecture design (i.e., focusing on the high-level structure of the software) and detailed design (i.e., focusing on the details of each component as well as its development).

(3) Software construction

Software construction means the implementation of the working software, which is based on the outputs from requirements engineering and software design.

(4) Software testing

Software testing aims to verify the software through testing, for example, whether the expected requirements are met and the expected behaviors are provided.

(5) Software maintenance and evolution

Software can change over time (e.g., caused by new requirements) and need to be maintained (e.g., correcting defects of the software) after delivering the software.

Appendix C. Results

References

- [1] M.S. Ali, M. Ali Babar, L. Chen, K.J. Stol, A systematic review of comparative evidence of aspect-oriented programming, *Inform. Softw. Technol.* 52 (9) (2010) 871–887.
- [2] L. Chen, M.A. Babar, H. Zhang, Towards an evidence-based understanding of electronic data sources, *Proceedings of the 14th International Conference on Evaluation and Assessment in Software Engineering (ESEM)*, Bolzano-Bozen, Italy, 2010, pp. 135–138.
- [3] V. Basili, G. Caldiera, D. Rombach, The goal question metric approach, *Encyclopedia of Software Engineering*, John Wiley & Sons, New York, 1994.
- [4] B. Kitchenham and S. Charters. *Guidelines for Performing Systematic Literature Reviews in Software Engineering*, Version 2.3. EBSE Technical Report EBSE-2007-01, Keele University and Durham University, 2007.
- [5] D. Garlan, R. Allen, J. Ockerbloom, Architectural mismatch: why reuse is still so hard, *IEEE Softw.* 26 (4) (2009) 66–69.
- [6] C. Wohlin, Guidelines for snowballing in systematic literature studies and a replication in software engineering, *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, London, UK, 2014, p. 38.
- [7] C. Yang, P. Liang, P. Avgeriou, A systematic mapping study on the combination of software architecture and agile development, *J. Syst. Softw.* 111 (1) (2016) 157–184.
- [8] R. Roeller, P. Lago, H. van Vliet, Recovering architectural assumptions, *J. Syst. Softw.* 79 (4) (2006) 552–573.
- [9] C.B. Haley, R.C. Laney, J.D. Moffett, B. Nuseibeh, Using trust assumptions with security requirements, *Requir. Eng.* 11 (2) (2006) 138–151.
- [10] M.M. Lehman, J.F. Ramil, Rules and tools for software evolution planning and management, *Annals Softw. Eng.* 11 (1) (2001) 15–44.
- [11] C. Yang, P. Liang, P. Avgeriou, A survey on software architectural assumptions, *J. Syst. Softw.* 113 (3) (2016) 362–380.
- [12] P. Bourque, R.E. Fairley, *Guide to the Software Engineering Body of Knowledge (SWEBOK (R))*: Version 3.0, IEEE Computer Society Press, 2014.
- [13] M.A.A. Mamun, J. Hansson, Review and challenges of assumptions in software development, *Proceedings of the 2nd Analytic Virtual Integration of Cyber-Physical Systems Workshop (AVICPS)*, Vienna, Austria, Article No. 2011, p. 7.
- [14] G.A. Lewis, T. Mahatham, L. Wraga, *Assumptions Management in Software Development*, CMU/SEI-2004-TN-021, 2004 Technical Report.
- [15] A. Steingruebl, G. Peterson, Software assumptions lead to preventable errors, *IEEE Secur. Privacy* 7 (4) (2009) 84–87.
- [16] A. Bazaz, J.D. Arthur, J.G. Tront, Modeling security vulnerabilities: a constraints and assumptions perspective, *Proceedings of the 2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing (DASC)*, Indianapolis, IN, USA, 2006, pp. 95–102.
- [17] B.G. Glaser, A.L. Strauss, *The Discovery of Grounded Theory: Strategies for Qualitative Research*, Aldine Publishing, New York, 1967.
- [18] F.J. Corbató, On building systems that will fail. *ACM Turing Award lectures* 34 (9) (1991) 72–81 Year Awarded 1990.
- [19] T. Dybå, T. Dingsøyr, Empirical studies of agile software development: a systematic review, *Inform. Softw. Technol.* 50 (9) (2008) 833–859.
- [20] K. Petersen, R. Feldt, S. Mujtaba, M. Mattsson, Systematic mapping studies in software engineering, *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, Bari, Italy, 2008, pp. 68–77.
- [21] S. Adolph, W. Hall, P. Kruchten, Using grounded theory to study the experience of software development, *Empirical Softw. Eng.* 16 (4) (2011) 487–513.
- [22] P. Lago, H. van Vliet, Observations from the recovery of a software product family, *Proceedings of the 3rd International Conference on Software Product Lines (SPLC)*, Boston, MA, USA, 2004, pp. 214–227.
- [23] C. Yang, P. Liang, P. Avgeriou, U. Eliasson, R. Haldal, P. Pelliccione, T. Bi, An industrial case study on an architectural assumption documentation framework, *J. Syst. Softw.* 134 (12) (2017) 190–210.
- [24] C. Yang, P. Liang, P. Avgeriou, U. Eliasson, R. Haldal, P. Pelliccione, Architectural assumptions and their management in industry – An exploratory study, *Proceedings of the 11th European Conference on Software Architecture (ECSA)*, Canterbury, UK, 2017, pp. 191–207.
- [25] P. Lago, H. van Vliet, Explicit assumptions enrich architectural models, *Proceedings of the 27th International Conference on Software Engineering (ICSE)*, St Louis, Missouri, USA, 2005, pp. 206–214.
- [26] D.V. Landuyt, E. Truyen, W. Joosen, Documenting early architectural assumptions in scenario-based requirements, *Proceedings of the 2012 Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA)*, Helsinki, Finland, 2012, pp. 329–333.
- [27] D.V. Landuyt, W. Joosen, Modularizing early architectural assumptions in scenario-based requirements, *Proceedings of the 17th International Conference on Fundamental Approaches to Software Engineering (FASE)*, Grenoble, France, 2014, pp. 170–184.
- [28] E. Denney, B. Fischer, A verification-driven approach to traceability and documentation for auto-generated mathematical software, *Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Auckland, New Zealand, 2009, pp. 560–564.
- [29] R. Seater, D. Jackson, R. Gheyi, Requirement progression in problem frames: deriving specifications from requirements, *Requir. Eng.* 12 (2) (2007) 77–102.
- [30] *ARIANE 5 Flight 501 Failure Report by the Inquiry Board*, <http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html>, (1996).
- [31] D. Giannakopoulou, C.S. Păsăreanu, H. Barringer, Component verification with automatically generated assumptions, *Autom. Softw. Eng.* 12 (3) (2005) 297–320.
- [32] J.M. Cobleigh, G.S. Avrunin, L.A. Clarke, Breaking up is hard to do: An evaluation of automated assume-guarantee reasoning, *ACM Trans. Softw. Eng. Methodol.* 17 (2) (2007) 7.
- [33] S. Chaki, E. Clarke, N. Sharygina, N. Sinha, Verification of evolving software via component substitutability analysis, *Formal Methods Syst. Des.* 32 (3) (2008) 235–266.
- [34] C. de la Riva, J. Tuya, Automatic generation of assumptions for modular verification of software specifications, *J. Syst. Softw.* 79 (9) (2005) 1324–1340.
- [35] C. Yang, P. Liang, Identifying and recording software architectural assumptions in agile development, *Proceedings of the 26th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, Vancouver, Canada, 2014, pp. 308–313.
- [36] A. Tang, Y. Jin, J. Han, A rationale-based architecture model for design traceability and reasoning, *J. Syst. Softw.* 80 (6) (2007) 918–934.
- [37] J. Misra, K.M. Chandy, Proofs of networks of processes, *IEEE Trans. Softw. Eng.* 7 (4) (1981) 417–426.
- [38] M. Abadi, L. Lamport, Conjoining specifications, *ACM Trans. Program. Lang. Syst.* 17 (1995) 507–534.
- [39] C. Yang, P. Liang, and P. Avgeriou. *Assumptions and their management in software development: a systematic mapping study – examples, classifications, and lessons learned*. <http://www.cs.rug.nl/search/uploads/Resources/AssumptionsSMS-ExamplesClassificationsandLessonsLearned.zip>.
- [40] H. Ziv, D. Richardson, R. Klösch, *The Uncertainty Principle in Software Engineering* (1997) Technical Report.
- [41] W.S. Humphrey, *A Discipline for Software Engineering*, SEI Series in Software Engineering, Addison-Wesley, 1995.
- [42] J.Y.C. Liu, H.G. Chen, C.C. Chen, T.S. Sheu, Relationships among interpersonal conflict, requirements uncertainty, and software project performance, *Int. J. Project Manag.* 29 (5) (2011) 547–556.
- [43] J. Whittle, P. Sawyer, N. Bencomo, B.H.C. Cheng, J.M. Bruel, RELAX: A language to address uncertainty in self-adaptive systems requirement, *Requir. Eng.* 15 (2) (2010) 177–196.
- [44] B.H.C. Cheng, P. Sawyer, N. Bencomo, J. Whittle, A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty, *Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, Denver, CO, USA, 2009, pp. 468–483.
- [45] P. Kroll, P. Kruchten, *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP* (2003).