Noname manuscript No.
(will be inserted by the editor)

A Systematic Literature Review on Trust in the Software Ecosystem

Fang Hou · Slinger Jansen

Received: date / Accepted: date

Abstract
The worldwide software ecosystem is a trust-rich part of the world. Throughout the software life cycle, software engineers, end-users, and other stakeholders collaboratively place their trust in major hubs in the ecosystem, such as package managers, repository services, and software components. However, as our reliance on software grows, this trust is frequently violated by bad actors and crippling vulnerabilities in the software supply chain.This study aims to define software trust in the worldwide SECO, that is, to determine what signifies a trustworthy system or actor. We conduct a systematic literature review on the concept of trust in the software ecosystem. We acknowledge that trust is something between two actors in the software ecosystem, and we examine what role trust plays in the relationships between end-users and (1) software products, (2) package managers, (3) software producing organizations, and (4) software engineers. Two major findings emerged from the systematic literature review. To begin, we define trust in the software ecosystem by examining the definition and characteristics of trust. Second, we provide a list of trust factors that can be used to assemble an overview of software trust. Trust is critical in the communication between actors in the worldwide software ecosystem, particularly regarding software selection and evaluation. With this comprehensive overview of trust, software engineering researchers have a new foundation to understand and use trust to create a reliable software ecosystem.

Keywords Software ecosystem · Software trust · Software package evaluation · Literature review

Fang Hou
Department of Information and Computer Science, Utrecht University, the Netherlands
E-mail: f.hou@uu.nl

Slinger Jansen
Department of Information and Computer Science, Utrecht University, the Netherlands
School of Engineering Science, Lappeenranta University of Technology, Finland
E-mail: slinger.jansen@uu.nl

1 Introduction

Software reuse enables software engineers to build software systems without starting from scratch. Sometimes, when building software, they are confronted with a choice: build it themselves or integrate a component from another software producing organization. If the choice is made to integrate a component, the second choice is a selection problem: which component should be selected? There are typically many options to select from, depending on the software engineer's perspective on which factors are more relevant in the selection process. Factors such as feature completeness, software component quality, and technology can all determine whether the software engineer selects a component. While researchers have called for more structured approaches to software selection (Farshidi, Jansen and Deldar 2021), the selection process is typically completely dependent on the context of the software packagers or engineers.

Software ecosystems (SECOs) are sets of actors that collaboratively serve a market for software and services, typically with an underlying technical platform (Jansen, Cusumano and Brinkkemper 2013). The union of all SECOs creates the worldwide SECO, i.e., all software producing organizations, software end-users, and actors. SECOs have *upstream flows* from end-users to software engineers, which contain, for example, money and data, as well as *downstream flows*, which contain, for example, source code and packages. The original worldwide SECO is a trust-rich part of the world. End-users expect certain trustworthiness when downloading apps from their favorite application store. For instance, when the App Store provides checksums, a technology to determine the authenticity of the received data for each deliverable; the end-users can ensure that the data they receive is the same data from the software engineer. The difference between trust and trustworthiness will be explained in Section 4.1.4.

However, more often than not, there is no sound basis for trust in the SECO hubs. Trust can be considered as founded and unfounded. There are more soft ways to create founded trust, e.g., ensuring that the software engineer has been a productive member of the SECO for a long time or ensuring that the software developed achieves established quality levels. Both soft and hard ways do not provide watertight guarantees, as the external environment can also impact trust, e.g., new attacks from the external world. According to Sonatype (2021), the number of supply chain attacks is dramatically increasing, posing significant security risks to both software producing organizations and end-users. For example, bad actors shift their attacks by inserting malicious code into the source code repositories or software packages to acquire a critical advantage of time, allowing malware to spread throughout the supply chain.

Standardization, processes, and models have been proposed to evaluate, select, and adopt software to address "trust erosion" from academic and industrial perspectives. For example, ISO/ISO 9126: 2001 and ISO/IEC 25010 provide standards to specify, measure, and evaluate the quality of systems and software; STRAM (Security, Trust, Resilience, and Agility Metrics) is a system-level trustworthiness metric framework to provide evidence in the measurement and quality of trustworthy systems (Cho et al. 2019); and ABCDE (Acceptance, Behavior, Constraint, Design, Extension) computes the trustworthy degree of the software component based on user feedback (Wang et al. 2019). Despite this, not all of them are widely used to select software in the industry. The primary reasons are: (1) *"It lacks a*

trust model that is comprehensive enough to support trust management in the component software system at different decision points for solving various trust-related issues" (Yan 2008); (2) *"Usually targeting mainly on quality"* (Li et al. 2021); (3) *"Most of the factors are abstract and lack clarity or guides that can be used to quantify trust factors"* (Li et al. 2021); (4) *"Current solutions do not address this problem of trust changing with time"* (Grandison and Sloman 2000); (5) *"System security issues must be addressed before an application is trusted"* (McKnight 2005), *"however, to date there have been no empirical studies identifying the relationship between security and trust"* (Goode et al. 2015).

In this context, we conduct a systematic literature review to understand software trust in the SECO better and summarize the key impact factors based on their frequency of mention. The purpose of the literature review presented here is to lay the groundwork for research on the design of trust assessment mechanisms that will place trust at the center of the selection process for software engineers. By analyzing the perception of trust in the literature review manuscripts, we provide a comprehensive overview of relevant trust concepts and present a brief overview of this study in Figure 1.

The rest of this research is structured as follows. Section 2 provides the background of SECO in four entities: actors, relationships, ecosystem services, and flows. It explains how SECOs can affect software trust. Section 3 presents the research method we used in this study, which works as an acquisition process to collect the software trust concept and trust factors considered by the software end-users during the software selection. Section 4 provides a definition of software trust as well as a discussion on the intrinsic and extrinsic trust factors in terms of software products, package managers, software producing organizations, and engineers, respectively. The results show that quality is the decisive factor for software trust, followed by code & structure, and security. Section 5 highlights the effect of trust factors on the software selection, validity, consideration, and challenges of this work, as well as our future work. It is difficult to develop a consistent set of criteria for assessing software trust; therefore, our view is that trust assessment should be synthetic and drawn on from different perspectives and channels. When collecting and sharing information, we need to exercise caution to protect individual's privacy and the data's objectivity. Additionally, collecting data on the trust of proprietary software may be challenging. Finally, in Section 6, we conclude that current research on software trust limited to the software itself is insufficient. Future research should extend to more hubs and actors in SECO, e.g., software packages, package managers, and software producers.

2 Background of Software Ecosystems

To understand what aspects of SECOs can affect software trust, we present here an overview of the flows in SECOs in which trust plays a role. We sketch the primary entities and stakeholders in the SECO in Figure 2, along with their simplified relationships. We extend the search process beyond just software or stakeholders to relevant search areas for specific entities, such as dependencies, components, packages, and package managers.

The worldwide SECO can be seen as all organizational entities that produce or use software, including the relationships and flows between them. Depending

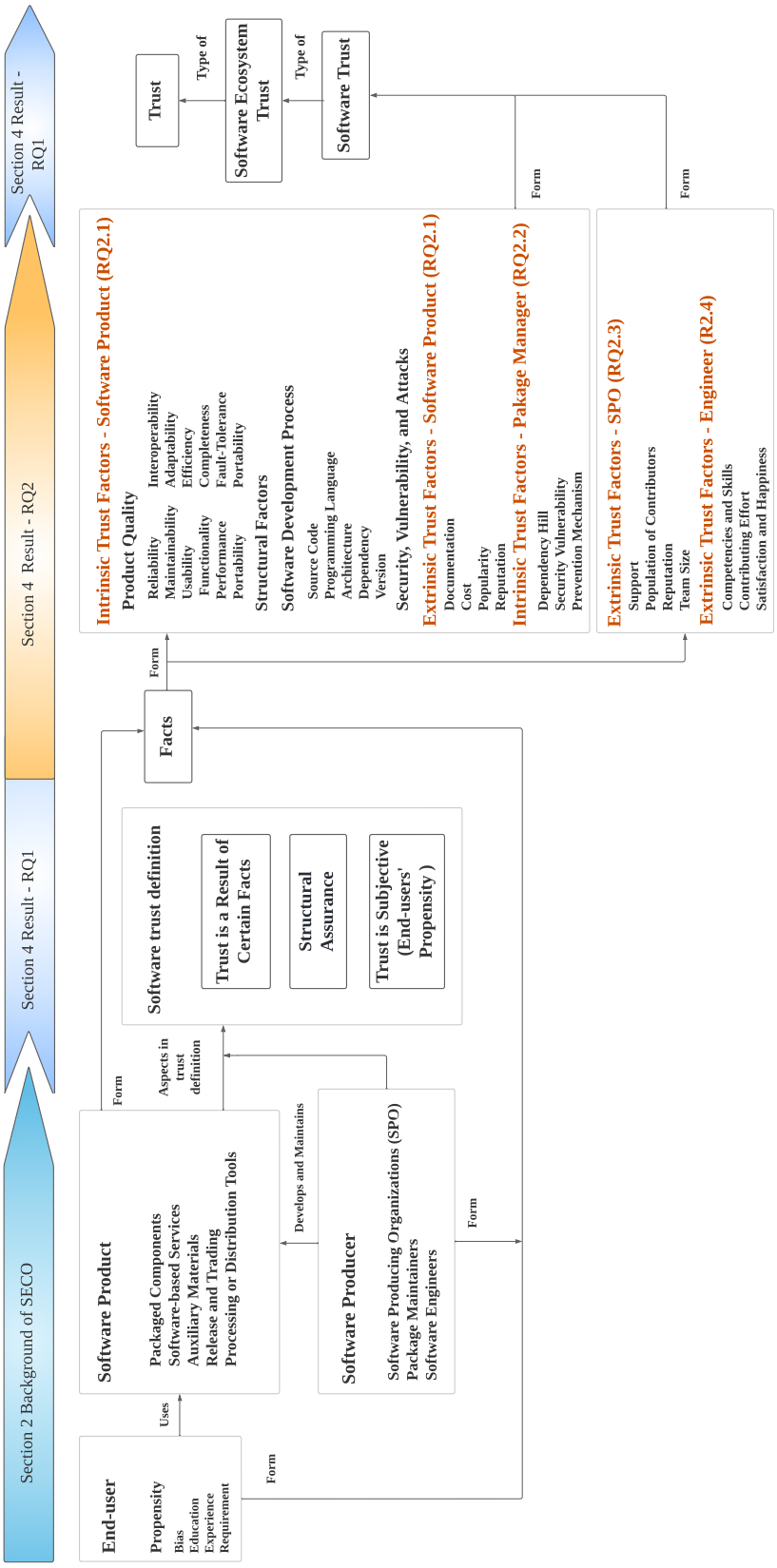

Figure 1 *This study outline provides the background of SECO as well as the main findings of this study, along with the associated section numbers.*

on the scope, one can place a specific perspective on the ecosystem, such as the "Twitter" ecosystem, which concerns all software producing organizations in its ecosystems and the end-users and end-user organizations that use the platform and associated products and services.

It is conceptually practical to distinguish four entities within SECOs: actors, relationships, ecosystem services, and flows. Ecosystem services are services that do not directly add any value to products but enable a better flow of products and services in an ecosystem, such as partner portals, application stores, and repository platforms. SECO flows are value flows across ecosystem relationships between two actors, such as products, knowledge, software, money, and data.

2.1 Actors in the Software Ecosystem

End-user A software end-user is an individual who adopts or intends to adopt a software product to make them more productive.
End-user Organization An end-user organization represents a set of end-users who collaboratively add software products to their software portfolio. They expect that the software products add value to their organization's goals. The distinction between an end-user and end-user organization is essential. This is because the decision process for adopting, for instance, a large resource planning application requires a large organizational selection process. In contrast, an end-user can easily choose to adopt, for example, a new text editor for daily use.
Software Engineer A software engineer is a talented individual who creates and maintains software products.
Software Producing Organization (SPO) An SPO is an organization that builds and maintains software to create valuable software that should be adopted as widely as possible (Jansen et al. 2012). SPOs typically employ software engineers who create and maintain software products. Generally they have additional goals, such as sustaining the organization through a business model.
Package Maintainers Package Maintainers are a subset of SPOs. Package maintainers are responsible for developing and maintaining software packages and their frameworks. They manage the development in code hosting platforms or repositories, such as GitHub. Additionally, they configure package builds to ensure that packages can be sourced from the distribution by extracting the source code from the collection of binaries in the distribution (Duan et al. 2021).

2.2 Flows in the Software Ecosystem

Software Product According to Xu and Brinkkemper (2007), a software product can be defined as follows: *"a packaged configuration of software components or a software-based service, with auxiliary materials released for and traded in a specific market"*. The definition emphasizes four elements: **packaged components** refer to software, packages, components, libraries, or codes; **software-based services** refer to the independent, small piece of functionality accessed through the internet; **auxiliary materials** refer to documentation, user manual or other materials which need to be implemented with the software or delivery to the end-users; and **release and trading** refer the activities that release the software to the market and related

commercial value. To this, we add another independent element: **distribution tools**, to emphasize channels that facilitate distribution downstream of the software supply chain, i.e., the systems or tools that assist in the processing or distribution of the packaged components that comprise a software product, e.g., package managers.

Component Based on Software Engineering Body Of Knowledge (SWEBOK), *"a software component is an independent unit, having well-defined interfaces and dependencies that can be composed and deployed independently"* (Bourque and R.E. Fairley 2014). The significant characteristic of a component is that it can be reused, interact with other objects, and be combined with other components to form a system or application. For instance, a menu class or a button class.

Library A library is a collection of prewritten functions or data structures that is organized to perform the same technically essential functions, such as functions that handle compatibility issues; it may be used as a dependency, avoiding the need to write code from scratch (Bauer, Heinemann and Deissenboeck 2012). For example, jQuery is a single JavaScript file, and it has been loaded with all of its dependencies to simplify various operations of JavaScript and resolve cross-browser compatibility issues. Sometimes a library refers to a package in programming language directives, such as npm, RubyGems, and Maven (Zerouali et al. 2018).

Package A package is simply a collection that contains software, libraries, and metadata. The metadata includes, for example, the software's name, its purpose, version number, providers, checksum, and a list of dependencies for the software or library version. Packages are in general versioned, which frequently adhere to de facto conventions, such as semantic versioning (Hanus 2018). A package distribution can be considered as a SECO, with a collection of interdependent software projects that are developed and maintained within the same environment (Decan and Mens 2019).

2.3 Ecosystem Services

Ecosystem services are services that enable better flows of value in the ecosystem, such as an application store in which end-user organizations can rapidly identify new solutions and add them to their application portfolios. While they do not add features and value to a product, they are of dire importance to make the products in an ecosystem successful. Ecosystem services can be provided by orchestrators, i.e., Google's Services for Android, such as Google Play. However, some ecosystem services are provided by third parties, such as GitHub and GitLab, who provide repository services to any actor in the worldwide SECO.

Package Manager A package manager provides a privileged, central mechanism for managing the installation and upgrade of packages on a computer's operating system automatically (Cappos et al. 2008a). Sometimes it refers to a software manager or application manager, depending on the context. In programming languages, the preferred term is package manager, because the installed software is often a set of libraries rather than a directly executable application. Package managers have a major contribution in that they retrieve the specific versions of the libraries required to build client applications and the libraries they depend on, and install the required libraries based on their dependencies (Hejderup, Deursen and Gousios 2018).

Package Repository Typically, a package repository is just a web server that hosts packages and their associated metadata (Cappos et al. 2008b). Package managers rely on package repositories to install packages and resolve dependency requirements.

2.4 Structure of the Software Ecosystem

In Figure 2 we provide the software package SECO, which is a workflow for package distribution as well. When an end-user uses a package manager to download a library, first, the package manager searches for files pointing to the repository location in the relevant configuration file it maintains, e.g., Apple Store or SourceForge, to retrieve the package. Subsequently, the package manager downloads the relevant packages from the repository. When the end-user confirms the installation of the selected package, the package manager downloads the package. A package may have dependencies, i.e., other packages must also be installed in sequence. The package manager can detect these dependencies and automatically download and install them in order. After a successful package installation, the package installation information is stored in the metadata of the local package database and managed by the package manager to maintain software dependencies and version information. End-users can modify the configuration file to retrieve packages from other repositories. Alternatively, they can search and download packages directly from repositories, such as GitHub.

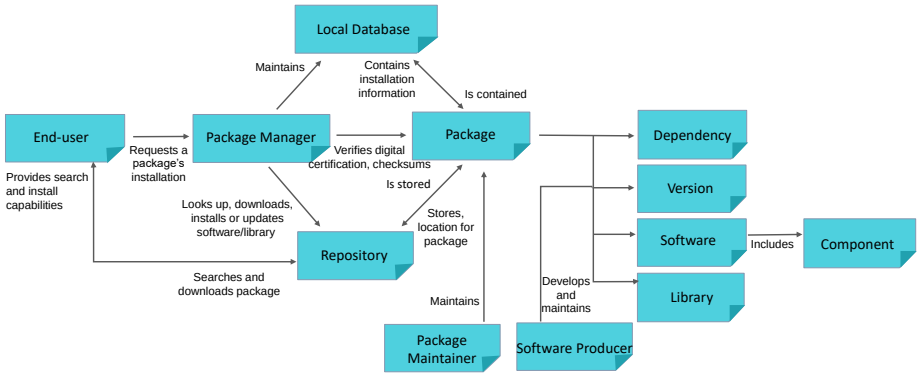

Figure 2 *This model shows the structure of the software package ecosystem. It identifies the prominent hubs and actors of SECO within the software selection process. These hubs and actors are the subjects of this study's discussion on trust.*

3 Research Method

Systematic Literature Review (SLR) is a term used to describe the process of collecting, reading, analyzing, refining, and organizing data in the existing literature to provide a comprehensive introduction, elaboration, and evaluation of a specific

research topic or phenomenon of interest (Keele et al. 2007). We performed this SLR following the guidelines and steps of Kitchenham (2004) to gain the current structure of knowledge on the trust within the SECO domain and to understand the current relationships between software end-users and software producers. We followed the outline of the following six activities in this review: (1) defining the research question; (2) searching for relevant manuscripts; (3) manuscript selection, including inclusion and exclusion criteria and quality assessment; (4) data extracting; and (5) coding scheme.

3.1 Research Questions

We construct the following research questions (RQ) to get an overview of software trust in worldwide SECO.

RQ1: How is the concept of software trust and SECO trust defined in literature?

Interest We try to study the concepts of software trust and SECO trust as well as the trust characteristics included in the concepts.

Approach This is achieved by searching the terms: "trust", "trustworthiness", "software trustworthiness", "software trust", "SECO trust", and "software ecosystem trust" from the selected manuscripts. We reviewed and compared relevant definitions and examined the sources, types, and attributions of software trust and SECO trust. Consequently, we extended the definition of software trust based on the relationships between software end-users and producers in the SECO and conclude our definition of software trust.

RQ2: What trust factors do end-user organizations consider when selecting software products?

RQ2.1: What trust factors do end-user organizations consider when selecting software products and versions?

RQ2.2: What trust factors do end-user organizations consider when selecting software package managers?

RQ2.3: What trust factors do end-user organizations consider when selecting software producing organizations?

RQ2.4: What trust factors do end-user organizations consider when selecting software engineers?

Interest We take into account end-user organizations' trust factors across the software supply chain. Each sub-questions observe a different link in the SECO, starting with the software product itself. Secondly, we identify the factors that play a role in selecting a software source, such as a package manager. Thirdly, we look at how end-user organizations perceive the SPO, and finally, we look at the software engineers that work for those organizations. In accordance with RQ1, we identify trust factors along different aspects and axes, with quantitative and qualitative ratings, that may positively or negatively affect perceived trust from the end-user organization.

Approach This is achieved by reading the full text of the literature review manuscripts and exploring the trust-related information from the given tables, models, frameworks, or context.

3.2 Search Strategy

Manuscripts were identified by using search strings from scientific libraries. The search strings are focused on three primary areas. To begin, we defined the scope of the study, including the context of SECO and software. Secondly, we discuss two specific aspects involved in SECO, namely concepts related to hubs, i.e., package, component, and dependency, as well as concepts related to software supply chain processes, i.e., software management, provenance, and developer. Thirdly, we incorporated frequent concepts related to trust based on the definition of trust, namely credibility, reputation, and uncertainty. This resulted in the following search string.

> (software ecosystem OR software) AND (package OR component OR dependency OR management OR provenance OR developer) AND (trust OR credibility OR reputation OR uncertainty)

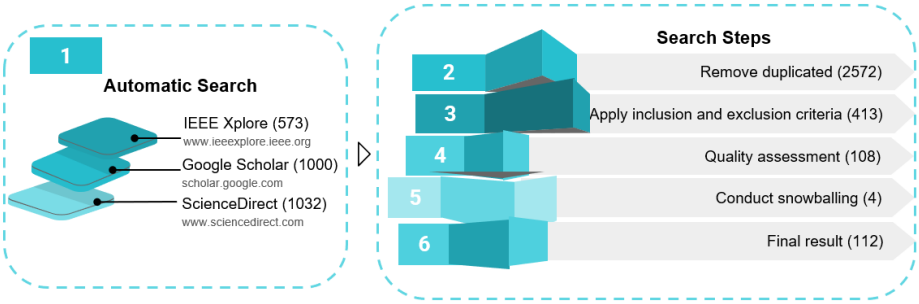

Figure 3 *This figure indicates the stages of the search process and the number of manuscripts at each stage.*

Following the guidelines of Petersen, Vakkalanka and Kuzniarz (2015), we designed the selection strategy in six steps. Figure 3 shows the selection process with the number of manuscripts included in each stage. These stages are described in more detail in the following subsections.

3.2.1 Stage1 - Automatic search

We adopted automatic search as the main search method and kept automatically selected manuscripts from IEEE Xplore (IEEE) and ScienceDirect as the academic sources and Google Scholar as a secondary source. We extracted 1000 manuscripts from Google Scholar, 1032 manuscripts from ScienceDirect, and 573 manuscripts from IEEE. And we stored the results in three lists.

3.2.2 Stage2 - Remove duplicates

Duplicate manuscripts (manuscripts with the same title, author, and year of publication in any two of the three sources) were existing in the results from the three lists. We manually removed 33 duplicate manuscripts, including 22 manuscripts from ScienceDirect and 11 manuscripts from IEEE, to ensure that the filtered list included unique manuscripts.

3.2.3 Stage3 - Apply inclusion and exclusion criteria

Inclusion and exclusion criteria ensure that relevant manuscripts are included, and ineligible manuscripts are excluded. If the inclusion criteria are too broad, poor-quality studies may be included, reducing the overall quality of the study results. On the contrary, if the inclusion criteria are too stringent, the resulting studies are likely to be small and therefore not generalizable (Meline 2006).
The given inclusion criteria adopted in this research are:

- Studies published since the 1990s;
- When studies reported the same research, only the most recent one was included;
- Studies that were in the field of software engineering.

The given exclusion criteria selected in this research are:

- Studies that were not accessible in full-text;
- Studies that were not peer-reviewed;
- Studies that were not presented in English;
- Studies that were incomplete, short papers, or only provided literature in the form of abstracts, prefaces, or presentation slides;
- Studies that were books or gray literature.

Based on the above criteria, we conducted two rounds of elimination. In the first round, the first author filtered the three lists by manuscript type, publication year, source (e.g., conference or journal name), and title. In the filtering of manuscript type and publication year, 57 manuscripts of patent type, 62 manuscripts of book type, seven manuscripts of citation type, and one manuscript published before the 1990s were excluded from the Google Scholar list. Afterward, the source was filtered to ensure that the topics of the manuscripts were only focused on software engineering. Five hundred seventy manuscripts were left for ScienceDirect, 573 manuscripts for IEEE, and 868 manuscripts for Google Scholar. More manuscripts have been filtered based on scanning the title to ensure that they are consistent with our research topic. A total of 40 manuscripts were retained in the ScienceDirect list, 77 manuscripts in the IEEE list, and 435 manuscripts in the Google Scholar list.

In the second round, we accessed the manuscripts and scanned the title and abstract of each manuscript. Five hundred fifty-six manuscripts were almost equally assigned to five researchers. By scanning the title and abstract of each manuscript, more manuscripts have been excluded, including 40 gray literature, 78 manuscripts that cannot be accessed, seven incomplete manuscripts, 13 short papers, and one that was not peer-reviewed.

At this stage, we discarded 139 manuscripts, leaving 413 manuscripts.

3.2.4 Stage4 - Quality Assessment

To eliminate concerns and develop consistent criteria for manuscript quality assessment, all researchers read and discussed the same four manuscripts before beginning the quality assessment. Our quality assessment focused on the following questions:

- Does the study address at least one research question?
- Is the study based on research or expert opinion?
- Is there a clear statement of the purpose of the study?
- Was the data collection rigorous?
- Are the findings of the study clear?

We employed the double data extraction method proposed by Buscemi et al. (2006) to assess the quality of each manuscript, i.e., two people in two rounds assessed the quality of each manuscript. The five researchers performed the first-round assessment. They read the full text and eliminated 265 manuscripts based on the above quality assessment criteria. Most eliminated manuscripts focused on the non-software engineering trust, trust in the non-software selection, or topics unrelated to trust and software choices.

The first author conducted a quality assessment of all the results derived from the researchers in the second round. Twenty-two manuscripts received varying quality assessment results. These 22 manuscripts included four manuscripts about specific software systems, such as mobile applications or enterprise resource planning, and 18 manuscripts about topics other than software, such as trust in hardware, economy, or IoT. Following a discussion of these distinctions, we arrived at 108 at this stage. Because this step involves human judgment, the threat cannot be eliminated.

3.2.5 Stage5 - Conduct Snowballing

The first author added four studies related to our topic through the snowballing technique. Since there were limited manuscripts on SECO in the search results, we used backward snowballing to check studies from the reference lists of SECO-related manuscripts with the keyword "trust". Unfortunately, only one was included (Schuur, Jansen and Brinkkemper 2011). However, from the reference lists, we still found three studies related to our study, including one manuscript on the software engineer's concerns about the use of library/framework in SECO (Haenni et al. 2013); one manuscript on a model for quality assurance of an ecosystem (Wang 2011); and one SLR on a comprehensive overview of SECO (Manikas and Hansen 2013).

3.2.6 Stage6 - Final result

Finally, we have 112 manuscripts used in data analysis.

3.3 Data Extraction and Synthesis

As explained earlier, for RQ1, we searched for terms including "trust", "trustworthiness", "software trustworthiness", "software trust", "SECO trust", and "software

ecosystem trust" in the selected manuscripts. The definitions and characteristics of software trust were usually explicitly given in these manuscripts. We provide sample definitions in Table 7, and more details will be discussed in Section 4.1. For RQ2, considerable impact factors were listed in tables, models, frameworks, or formulas. There are still a number of factors that were discussed in context in some manuscripts. There may be ambiguity surrounding these factors, or they were difficult to detect or were not even considered a factor. Thus, the difficulty in data extraction lies at this point.

To address this challenge, we adopted the qualitative data analysis approach for data extraction, which was carried out by five researchers in two rounds. In the first round, all researchers performed data extraction for the same four manuscripts. Consensus meetings were held to ensure a shared understanding of what data should be extracted, which is listed in Table 1. Then five researchers studied the different manuscripts and recorded the information in one excel sheet. After reviewing and confirming all extractions from the second round, the first author classified the data, which was verified by the second author.

Table 1 *A summary of the attributes that have been extracted from each manuscript during the data extraction stage. The attributes cover the basic information (title, author(s), publication year, source), topic (keywords and SECO type), research method, and information related to our research (answered RQ, given trust definition(s) and trust factor(s)) of a manuscript.*

Attribute	**Description**
Title	The title of the manuscript
Authors	The author(s) of the manuscript
Source	Source of the selected manuscripts, e.g., conference or journal
Year	The year of the publication
Keywords	Keywords of the manuscript
Research method	The research method of the manuscript
SECO type	Type of the SECO
Research questions	Which research question(s) has(have) been answered
Trust definitions	Trust definition(s) given in the manuscript
Trust factors	Main identified trust factor(s)

Data synthesis was performed using a frequency analysis technique to aggregate the extracted data, calculated based on the total number of manuscripts in which the words or synonyms appear. All included manuscripts were analyzed and extracted. It should be noted that because some models or criteria have been proposed based on the results of previous research, several trust factors are duplicated. However, we still counted and accumulated these trust factors. The reason is that we believe the trust factors that are frequently discussed must be significant.

We focused on software trust in software selection, but software and trust are broad concepts from which considerable relevant topics can be deduced and developed. Therefore, the search results encompassed a broad range of topics. We assigned those topics to distinct sub-RQs based on their similarity and relevance, as outlined in Table 2.

Table 2 Main research areas per research question. The topic area is determined by the title, abstract, or keywords of the selected manuscripts. Interest shows the aspects that we focus on according to the different research questions.

Topic Area	**Topic of Interest**	**RQ**
Software trust Software trustworthiness Software ecosystem trust SECO trust Trust Trustworthiness	Definition and attributes of software trust and SECO trust	RQ1
Application Project Free open source software (FOSS) Open source software (OSS) OSS component Packaged software Software engineering Third-party software Software-as-a-Service (SaaS) Product Commercial off-the-shelf software (COTS) Software SECO OSS Ecosystem Software service Information system Internet-based inter-organizational system (IIOS)	Software evaluation and selection criteria Software metrics and reliability Relationship between end-users and technology	RQ2.1,2.3,2.4
Library OSS code repository Component Package	Evaluation and selection criteria	RQ2.1
Package manager Package management system	Evaluation and selection criteria	RQ2.2
Software provider Supplier Software organization Software company Vendor Package provider SaaS provider OSS provider OSS community	Relationship between end-users and SPOs Impact factors of SPOs and stakeholders	RQ2.3
Developer Software engineer IT staff Maintainer	Relationship between end-users and software engineer Impact factors of software engineers	RQ2.4
Security Vulnerability Attack Uncertainty Risk management	Risk, attack or uncertainty in software, packages and managers	RQ2.1,2.2

3.4 Coding Scheme

The coding scheme was developed based on the data extraction results. During the coding process, we found four common situations. First, some of the manuscripts used the same codes, such as cost, reputation; or similar codes such as functionality (Godse and Mulik 2009), or function (Jadhav and Sonar 2009); second, the same concept adopted different names, e.g., vendor (Del Bianco et al. 2011), supplier (Pollock and Williams 2007), software provider (Hillebrand and Coetzee 2013), and software producer (Wang et al. 2019); third, some trust factors were not be codified but could be broken down in some chunks, e.g., advice given by other concerned parties (Chau 1994); and fourth, codes were usually located in different categories in different manuscripts. For example, active maintenance is located under technical factors as it is considered a major activity of the software release process (Vargas et al. 2020). However, Jadhav and Sonar (2009) consider it as an organizational factor, since software maintenance is performed by the organization.

Hence, we adopted an inductive coding method to analyze the data from each manuscript to identify the relationships between them, and classify the trust impact factors with Cue Utilization Theory (CUT). Inductive coding is a type of data analysis in which the researcher reads and interprets raw textual data to develop concepts, themes or a process model based on the data (Chandra and Shang 2019). CUT is a well-known framework for understanding and analyzing various factors that influence a topic, as well as evaluating a product (Midha and Palvia 2012). We classified the "cues" extracted from the selected manuscripts using CUT and specified the characteristics of each category to ensure that they can provide a comprehensive picture of the trust factors. We then divided those factors into intrinsic and extrinsic trust factors for the previously proposed research questions. Intrinsic trust factors have been used to represent a product's physical attributes. In this study, they refer to source code or architecture-related attributes or factors, such as quality-related factors or vulnerabilities. Extrinsic trust factors have been used to represent external attributes, such as the product's reputation, cost, licenses, or capability of the software producer. In addition, in the context of the SECO, each actor in SECO and its relationships, for example, the reputation or popularity of software producers, are also categorized as extrinsic trust factors.

Once we had a high-level classification of trust factors, we combined the RQs to derive subcategories. RQ2.1 and 2.2 could be classified as both intrinsic and extrinsic factors, while RQ2.3 and 2.4 were only classified as extrinsic factors. For example, we recognized "active maintenance" as a meaningful activity in the software release process that answers RQ2.1 - trust factors of software products, which focuses on whether end-users receive regular or timely active maintenance in software releases. Therefore it is coded under the intrinsic technical factors. However, considerable end-users also focus on the producers' inputs, e.g., support or service, operation, and maintenance, which is more inclined to the quality of support the software producers provide. Hence we coded "support or service" as a sub-factor to answer RQ2.4.

After conducting a systematic review of all trust impact factors, we assigned them to our CUT categories based on the research questions. We then labeled the codes using the words from manuscripts and their CUT and semantic counterparts and adapted and refined them. Additionally, we determined each category's relative importance by counting the words' frequency. If a trust factor appears frequently,

it indicates that it is significant in research on software trust and may have a significant impact. In Table 9 we show the major trust factor according to the word frequency. To ensure traceability, we provided the coding scheme[1] in an open repository.

3.5 Biases in the SLR Process

The challenge of bias is one that all researchers face. The reasons may be, for example, that a researcher has insufficient understanding of the findings of a subject, perhaps awares but unable to access the findings, or the findings may be missing critical data points. Bias may result in missing data, making not only the collected sample size decrease but also the validity of the sample smaller (Cooper, Lindsay and Patall 2008). In this study, we faced several biases: **Publication bias** refers to the problem that positive results tend to be published more frequently than negative results. If negative results are not widely published enough, false conclusions could be attributed to them as true (Keele et al. 2007). **Retrieval bias** refers to the risk that the sample used in the synthesis does not appropriately represent the literature available (Durach, Kembro and Wieland 2017). To mitigate them, we conducted an automatic search. Search strings were not limited to only certain qualities of software products, but a broader concept of software trust. In addition, we added studies by backward snowballing to provide a more comprehensive search. **Biases in extractions and synthesis** caused us to be unable to extract comprehensive trust data from all manuscripts (Drucker, Fleming and Chan 2016). Due to our different backgrounds, researchers have varying understanding of SECO, trust, and even a seemingly basic concept. We have attempted to reduce these biases by having two researchers extract data from the same manuscript and reach a consensus through discussion. However, human judgment is inherently subjective and biased, and the threat is difficult to eliminate. It caused some potential studies or factors could have been missed. To address it, we need to add to the studies afterward to adjust the trust factors at any time to make the study more comprehensive.

3.6 Replication Package

This study is accompanied by the data set[2] that contains all manuscripts at the different stages of selection in this literature review.

4 Results

This section reports our findings on end-user organizations' trust factors. First, we will present the extraction results, and the remaining parts are organized according to the research questions.

Table 3 shows the number of publication types. The most frequent manuscripts are from journals. The research methods are classified according to the categories

[1] https://data.mendeley.com/api/datasets/xn4jk93g4t/draft/files/b36ddc93-0f42-4227-b966-4c07d2a7e429?a=9c51b45f-c943-4a22-814f-5baf59fb2896

[2] https://figshare.com/s/dadba2e7b3a6ab71ef18

listed in Table 4. The majority of the selected manuscripts are empirical research. Another large portion of the trust factors can be attributed to theories, models, or frameworks that analyze or evaluate software trust or specific aspects of software products. The remaining factors come from exploratory research. The methods used to collect trust data are shown in Table 5. The majority of the trust factors or trust facts are the result of experiments. These are approaches, models, or frameworks proposed in the literature.

Table 3 Summary of Publications

Source	Total
Journal	79
Conference	26
Workshop	7

Table 4 Summary of Research Methods

Research Methods	Total
Empirical	50
Theoretical	45
Exploratory	16
Commentary	1

Table 5 Summary of Data Collection Methods

Collection Methods	Total
Experiment	45
Survey/Interview	21
Case Study	18
Documents	15
Literature Review	11
User study	1
Commentary	1

The numbers of search results for each research question are given in Table 6. Software components make up the most significant proportion of the selected manuscripts, followed by manuscripts on software development and software trust definition. The manuscripts on the package managers and software packages make up the least amount of literature. Despite adding "package" to the search strings, the number of retrieved manuscripts is limited regarding software package trust and its related terms.

Table 6 The Number of Manuscripts per Research Question

Research Question	Total
RQ1:How is the concept of software trust and SECO trust defined in literature?	35
RQ2.1:What trust factors do end-user organizations consider when selecting software products and versions?	91
RQ2.2:What trust factors do end-user organizations consider when selecting software package managers?	4
RQ2.3:What trust factors do end-user organizations consider when selecting software producing organizations?	18
RQ2.4:What trust factors do end-user organizations consider when selecting software engineers?	19

Note: A manuscript can address more than one research question, so the total number of manuscripts is not 112.

4.1 RQ1 - Concept of Software/SECO Trust in Literature

In this subsection, we provide an overview of the general field of software trust to answer RQ1: *How is the concept of software trust and SECO trust defined in literature?* A total of 39 definitions of software trust can be found in the manuscripts.

Table 7 shows some examples of those definitions. Unfortunately, we did not find a specific definition of SECO trust. We did not find a common definition of software trust or trustworthiness, because it varies depending on the context (Heiskanen, Newman and Eklin 2008). And trust is often replaced by synonyms, such as **reliability**, **dependability**, or **assurance**.

Table 7 The Number of Manuscripts per Trust Definition

Definition	Total	Example definition
Software trustworthiness	11	"The trustworthiness of software can be simply defined as the degree of confidence that exists that it meets a set of requirements" (Immonen and Palviainen 2007).
Common themes trust	8	"Trust is the willingness of the trustor (evaluator) to take the risk based on a subjective belief that a trustee (evaluatee) will exhibit reliable behavior to maximize the trustor's interest under uncertainty (e.g., ambiguity due to conflicting evidence or ignorance caused by complete lack of evidence) of a given situation based on the cognitive assessment of experience with the trustee" (Cho, Chan and Adali 2015).
System trust	8	"System trust, refers to the belief that the appropriate impersonal structures are in place to allow one party to anticipate successful transactions with another party" (Roumani, Nwankpa and Roumani 2017).
Software service trust	5	"Trust is the individual's perspective on a particular service, or product and reputation is a group's perspective on a particular service or product" (Hillebrand and Coetzee 2013).
Software trust	4	"Trust is viewed as an attribute of software that combines concerns for reducing the potential for both innocent errors and malicious insertions" (Amoroso et al. 1991).
Third-party library trust	2	"The first type of trust is a library's functional and non-functional related specifications. The second type of trust is that the introduced library is not volatile towards the current system environment" (Kula et al. 2015).
Internet application trust	1	"We define trust as "the firm belief in the competence of an entity to act dependably, securely and reliably within a specified context" (Grandison and Sloman 2000).

4.1.1 Trust is Subjective

Trust is subjective. According to Del Bianco et al. (2011), trust is a kind of "confidence"; Guo et al. (2014) state that trust is a "belief"; Lai, Tong and Lai (2011) consider is as an "expectation"; Cho, Chan and Adali (2015) explain it as a "willingness"; Das and Teng (2001) describe it as a "goodwill". Yan (2008) argues that trust is a propended and subjective conclusion that people draw based on a set of information. The propensity of software end-users depends on their experience, bias, or education (Amoroso et al. 1991; Cho, Chan and Adali 2015). For example, a knowledgeable end-user can determine whether a software package is trustworthy and appropriate for the current project based on his previous experience. Correspondingly, a person without any IT background and experience has little propensity in the software package selection.

The requirement is another kind of propensity because the *"trustworthiness requirements depend on how, why, where, and by whom the software is used"* (Del Bianco et al. 2011). A software product may not give the same confidence level across all functions but remains dependable (Jackson 2009). For example, a bank system may incorrectly calculate the expiration date of a credit card but must not disclose the customer's information; or a cellphone may be unable to change the ringtone, but it can make a call. Therefore, software trust has boundaries, which encompass the specific functionality, timing, user roles, experience, and requirements of the software (Jackson 2009).

Such propensity is temporal and may change depending on the end-user's knowledge, experience, and different requirements for the software (Moyano, Fernandez-Gago and Lopez 2016). Amoroso et al. (1991) point out that it also can be eliminated by providing specific and detailed guidelines.

Table 8 The Number of Software Product Attributes in the Trust Definition

Definition	**Total**
Quality	8
Security	8
Reliability	7
Behavior or performance	7
Service	5
Risk and uncertainty	4
Dependability	3
Reputation	3
Resilience and agility	3

4.1.2 Trust is a result of Certain Facts

Software trust is inherently predictive based on a set of historical data, regardless of its focus (Alarcon et al. 2020; Jackson 2009; Jadhav and Sonar 2011; Kula et al. 2015; Mcknight et al. 2011). According to this interpretation, it can be found in those given definitions that trust in software is established through the behavior or performance of the software producer and the software. Table 8 shows the attributes most often mentioned in trust-related concepts.

Quality Trustors generate their trust based on a number of observed behaviors or performances, where qualitative attributes are primary (Yan 2008). The software must be of high-quality (Hillebrand and Coetzee 2013) and meet the trustor's standards (Yan 2008). Kula et al. (2015) explain such standards could be functional and non-functional. In addition, trust definitions explain certain quality characteristics, for example, "functionality, reliability, and helpfulness" (Mcknight et al. 2011); "fault tolerance, security, survival, and real-time capability" (Zhu et al. 2012); "predictability, and safety" (Cho et al. 2019); "availability and supportability" (Hong, Chang-hui and Ben 2011). They reflect the multifaceted quality requirements of end-users in various contexts.

Security Software Trustworthiness reflects the probability that a software product will experience a security failure in a given time (Bugiel, Davi and Schulz 2011). Security refers to preventing deliberate or accidental attacks (Boyes et al. 2014).

The term "software security failure" describes a flaw discovered by its software provider or the security community (Bugiel, Davi and Schulz 2011). Security failures can be caused by a variety of factors, both internal and external. Software trustworthiness varies depending on the end-user's environment and the standards of security (Bugiel, Davi and Schulz 2011). To improve software security and trust, **resilience and agility** as emerging concepts have been discussed in recent years in the study of software trust (Boyes et al. 2014; Cho et al. 2019). According to the researchers, they reflect the requirements for the ability of a software product to continuously restore a normal, functional system state and respond to unexpected changes or situations.

Risk and uncertainty It is common to find both risk and uncertainty in a definition of trust, as it is believed that the risk arises from uncertainty. Grodzinsky, Miller and Wolf (2011) believe that the fewer information end-users receive regarding a software product, the greater the uncertainty, the higher the risk they are exposed to. Similarly, Cho, Chan and Adali (2015) point out that uncertainty may result from ambiguity due to conflicting facts, or ignorance due to a complete lack of evidence. The presence of risk and uncertainty makes trust more meaningful. That is, despite some uncertainties and risks, end-users still hold positive expectations, i.e., trust, in the software product (Das and Teng 2001). It is possible to achieve software trust through risk management, for example, by putting in place appropriate personnel, physical, procedural, and technical controls (Boyes et al. 2014).

Reputation Trust and reputation are also topics that have been discussed in the literature. Trust refers to the perception that an individual holds of a service or product, whereas reputation refers to the perception that is held by a group (Hillebrand and Coetzee 2013). Similarly, Hoxmeier (2000) argues that *"reputation is built by satisfying market signals"*, it can be viewed as a complement to trust, i.e., the public's perception of the trustworthiness of the organization.

Mcknight et al. (2011) point out that people trust the software producer, not the software itself. Thus, the choice of the software product is a reflection of the end-user's trust in the reputation of the software producer. The reputation of software producers is obtained by examining evidence, signs, or their experience (Cho, Chan and Adali 2015), to identify whether they are considered professional, reliable, and capable of producing high-quality software products and providing quality services (Jadhav and Sonar 2011). Additionally, how software producers remediate and recover from risks and problems is also considered to be a factor that impacts the reputation of software producers as it affects the willingness of the end-user to maintain a long-term service relationship with the software producer (Bennett et al. 2000).

4.1.3 Structural Assurance

Structural assurance (SA) is a special kind of trust we found in the selected manuscripts. This kind of trust is not based on the attributes of the software product or its producer but is built by the end-user or the software producer's fear of the consequences of violating a rule, law, or contract (Cho, Chan and Adali 2015). As a result, SA ensures the interests of all stakeholders are not compromised by unclear assumptions, premises, and responsibilities, and contracts or clauses with unclear penalties.

4.1.4 Trust vs Trustworthiness

Trustworthiness is often misinterpreted as trust (Wright 2010). Trust is usually from the trustor's point of view. It is typically a belief that the trustee would do what is expected of the trustor (Guo et al. 2014; Jadhav and Sonar 2011; Mcknight et al. 2011). Trustworthiness is viewed from the perspective of the trustee. It is the probability that the trustee will act by the trustor's expectations (Bauer 2019). The trustworthiness of a trustee and the trustor's trust may differ. Trustworthiness can be seen as a characteristic of trust (Becerra, Lunnan and Huemer 2008), which means that trust cannot exist without a high level of trustworthiness (Heyns and Rothmann 2015).

4.1.5 Definition of SECO Trust

SECO is a broad and complex concept. It is not only a complex network of overlapping supply chains of software, data, and services but also a subtype of a business ecosystem. As such it inherits some properties of a business ecosystem, e.g., **business**, **strategy**, and **network**. As explained by Van Den Berk, Jansen and Luinenburg (2010), "business" implies not only profit or income but also includes possible benefits other than financial income; "strategy" includes SECO vision and platform's strategy; and "network" refers to the connection between SECOs, a more connected ecosystem is more resistant to shocks.

SECO trust is also a broader concept corresponding to the concept of SECO. As introduced in Section 1, it constitutes all upstream and downstream trust between the stakeholders that comprise the SECO. This includes, for instance, the trust of end-users in software engineers to build solutions for their (business) problems, and the trust of software engineers in software package management systems to transfer their software to end-users correctly. Moreover, it is influenced by business systems thinking and considers business, strategy, and networks. Also, SECO is unique because operational knowledge is distributed among the actors, and no one has a complete picture. Therefore, from a social perspective, the propagation of software operation knowledge in actors can increase the attractiveness of SECO and build trust (Schuur, Jansen and Brinkkemper 2011).

We did not find a specific definition of SECO trust, and we believe that the study of SECO trust should start with the trust in entities of SECO (as we presented in Section 2) and extend to the trust in the market and society, as well as in networks with worldwide ecosystems.

4.1.6 Definition of Software Trust

Software trust, which is a subsidiary of SECO trust, which is defined based on the complex relationships between software end-users and producers in the SECO, including (1) trust in hubs (e.g., dependencies, software packages, or package managers); and (2) trust in actors (e.g., software engineers, software producing organizations, or communities) in the ecosystem. The success of SECO depends upon software trust because in the absence of software trust, SECO cannot effectively deliver value to its stakeholders. Considering the context of SECO, we summarize the definition of software trust as follows:

Software trust refers to the willingness of SECO actors to accept risks based on subjective beliefs. It is essentially an upstream trust in that actors expect assurance that other actors on top of a common technology platform can exhibit reliable behavior and provide valuable software products. Software is expected to be able to carry out their intended functions in the presence of uncertainty and run consistently and reliably without interruption.

4.2 RQ2 - Trust Factors in software products selection

In the following sections, we discuss trust factors in each sub-research question. Table 9 shows the entire landscape of the factors collected from the selected manuscripts. We focus on frequently occuring trust factors, that are mentioned less frequently are not discussed here.

4.2.1 RQ2.1 - Trust Factors of Software Products

This subsection argues *RQ2.1 - What trust factors do end-user organizations consider when selecting software products and versions?* As discussed in Section 3.4, based on CUT, we classified the trust factors as intrinsic and extrinsic factors. Intrinsic trust factors refer to the software product's features, code, and architecture, such as software quality, code quality, or known vulnerabilities. Extrinsic trust factors refer to attributes bestowed on the software product by the outside world, such as popularity, reputation, software-based services, or auxiliary materials that need to be delivered to the end-users. We highlighted the major factors as **boldface** in the subsequent sections.

Intrinsic Trust Factor

Intrinsic trust factors represent internal physical attributes rather than being determined by the environment. The following factors are discussed frequently in the manuscripts:

Security, Vulnerability, and Attack Proneness Security is one of the primary software trust attributes. Surprisingly, even though software engineers are aware that it is a significant factor, they affirm that they do not check it in advance before adopting a software product (Vargas et al. 2020). The authors report that software engineers do not check everything but just utilize community knowledge, for instance, publicly accessible vulnerability databases, to gather data. However, those public vulnerability databases are not a perfect solution, as they face two challenges. The first one is the diversity of the data source. According to Decan, Mens and Constantinou (2018), there are three sources of vulnerability data: (1) public databases, such as CVE, NVD, or snyk.io; (2) specialized tools, such as the GitHub vulnerability tracking mechanism; (3) mailing lists, bug tracking systems, and blogs. The authors point out that the diversity of vulnerability data sources makes it impossible to fully understand the vulnerabilities of an ecosystem. The second one is the timeliness of vulnerability reporting. The quality of the data in the vulnerability database impact vulnerability tracking, analysis, and fixing. With the emergence of new types of cyberattacks, software attacks show a gradual trend of a large-scale multivector designed to infect multiple components of the SECO.

Table 9 This table identifies the main trust factors of RQ2 along with the number of times each factor was mentioned. The most frequently discussed factor is quality. However, to present more detailed trust factors, we did not list quality as a separate trust factor, but rather its subfactors.

Intrinsic Factors - Software Product (RQ2.1)	
Source code	31
Reliability	29
Security	29
Maintainability	18
Process	18
Usability	16
Vulnerability and attack	13
Functionality	13
Framework	10
Performance	10
Dependency	8
Portability	8
Safety	7
Interoperability	7
Adaptability	6
Efficiency	5
Completeness	5
Fault-Tolerance	5
Portability	5
Extrinsic Factors - Software Product (RQ2.1)	
Documentation	24
Structural assurance	22
Cost	17
Popularity	11
Reputation	7
Intrinsic Factors - Package Manager (RQ2.2)	
Dependency	4
Security vulnerability	4
Prevention mechanism	2
Extrinsic Factors - Software Producing Organization (RQ2.3)	
Support	5
Population of contributors	5
Reputation	4
Team Size	2
Extrinsic Factors - Software Engineer (RQ2.4)	
Competencies and skills	5
Contributing effort	2
Satisfaction and happiness	2

Hence, it is difficult for software engineers or end-users to discover and report all vulnerabilities from all hubs promptly (Duan et al. 2021).

In addition, there are problems faced in fixing vulnerabilities. In most cases, fixing them can take a long time. An empirical study of nearly 400 security reports in npm dependencies over six years showed that only half of the vulnerabilities could be fixed within a month, and the probability of a vulnerability being fixed after six months is 74% (Decan, Mens and Constantinou 2018). This study also indicates that more than half of the dependent packages are affected by vulnerabilities in upstream packages. Interestingly, due to the lack of proper maintenance of packages or strict dependencies, most affected dependent packages are not automatically fixed with the upstream packages. Therefore, Duan et al. (2021) believe that the

defense approaches that are more oriented toward preventing access to sensitive data or finding bugs in software products cannot effectively protect against attacks on the SECO; future research should focus on analyzing and detecting malicious software packages through an analysis of relationships between stakeholders in the SECO to prevent supply chain attacks.

Furthermore, software security still has no uniform standard to measure it (Wang et al. 2015a). This may be because software security is influenced by the factors, such as environment, complex software supply chains, project management, or the adoption of programming languages. For the software itself, the complexity of the requirements, the complexity of the code, the technology used, and even the skills of the engineers have an impact on software security, with the problem being that these factors are difficult to quantify. Meanwhile, it is unrealistic to request software testers to meet all the security requirements. Different stakeholders have different perceptions of software security. Contrary to organizations and governments, software engineers do not place software security as a priority (Vargas et al. 2020). In particular, software security requires significant resources to develop, evaluate, and defend; small and medium-sized companies may be unable to afford this investment.

Quality and Development Process Factors regarding quality revolve around the quality characteristics of the software itself and the activities of the software development life cycle (SDLC).

Software quality models contribute the majority of factors. Berander et al. (2005) introduce several well-known software quality models. McCall's Quality Model is proposed in 1977 and it is one of the more renowned predecessors of today's quality models, which includes 11 factors that describe the external view of the software (user view) and 23 factors that describe the internal view of the software (developer view). In the following year, Boehm's Quality Model is introduced with a hierarchical quality model structure. It focuses on a broader range of characteristics of software products, i.e., as-is utility, maintainability, and portability. FURPS & FURPS+ are developed by Robert Grady of Hewlett-Packard, identifying software quality factors through functional requirements and non-functional requirements. Dromey's Quality Model recognizes that quality is assessed differently for each product and therefore requires a combination of product characteristics and quality attributes to dynamically determine how each attribute affects quality attributes and identify software shortcomings. ISO/IEC 9126:2001 covers six significant areas of software evaluation: functionality, reliability, efficiency, maintainability, portability, and usability. However, there are no indications on how to quantify these factors and subfactors, and this model may reflect the perception of software engineers instead of end-users (Challa et al. 2011). It has been replaced by ISO/IEC 25010, which focuses on the characteristics of software internal quality, external quality, and usage quality. Notably, ISO/IEC 25010 proposes that trust is "the degree to which a user or other stakeholder has confidence that a product or system will behave as intended" in quality of use. Several studies have extended based on these well-known models and attempted to quantify the factors that affect software quality, such as the Fuzzy Software Quality Quantification Tool (FSQQT), which takes several real-time values of metrics from the ISO/IEC 9126 Model as inputs and quantifies software quality based on the views of end-users, engineers and project managers as outputs. This study shows that end-users care more about reliability and usability; engineers focus more on functionality, efficiency, portability,

Table 10 The tables show the primary factors, subfactors, and counts based on McCall's Model, Boehm's Quality Model, Dromey's Quality Model, ISO/IEC 9126:2001, ISO/IEC 25010, FURPS & FURPS+, and FSQQT Model.

Primary Factor	Total
Reliability	6
Efficiency	5
Portability	5
Usability	5
Functionality	4
Maintainability	4
Correctness	2
Testability	2

Subfactor	Total
Accuracy	5
Testability	4
Adaptability	4
Interoperability	3
Security	3
Maturity	3
Fault tolerance	3
Operability	3
Installability	3
Replaceability	3
Reusability	3
Consistency	3
Completeness	2

and maintainability; and project managers are more concerned with cost, schedule pressure, and cycle time (Challa et al. 2011).

Table 10 shows the number of primary factors and subfactors stated in the models of McCall's Model, Boehm's Quality Model, Dromey's Quality Model, ISO/IEC 9126:2001, ISO/IEC 25010, FURPS & FURPS+, and FSQQT Model. We found that reliability is the most critical factor in quality models. Efficiency, portability, usability, accuracy, testability, and adaptability are the most stated subfactors.

Software quality is not accidental. It is achieved through management and must be assured throughout the product life cycle. It is suggested by models or standards that evidence of software quality or software trust should cover the information obtained at each stage of the development process. For instance, IEEE Std 982.2-1988 indicates the relationship between software reliability and each life cycle phase, including concept, requirement, design, implementation, test, installation and checkout, operation and maintenance, and retirement. Even though this standard was published in 1988, it does, to some extent, reflect the information at each stage of the SDLC that influences the quality measurement. Additionally, a number of Capability Maturity Models (CMM) address quality issues from five maturity levels of software development processes, i.e., initial level, repeatable level, defined level, managed level, and optimizing level (Berander et al. 2005; Ellison, Nichols and Woody 2016). The repeatable level focuses on project management, the defined level focuses on the engineering process, the managed level focuses on product and process quality, and the optimizing level focuses on continuous improvement. He et al. (2009) argue that trusted components are affected by the development process. Therefore, trustworthy proof should be collected through three phases: development, submission, and application. Submissions and applications are primarily concerned with the quality factors in the quality models. For instance, the submission phase includes the proof of functionality, reliability, and usability, and the application phase emphasizes the proof of effectiveness, security, and efficiency. Similarly, Wang (2011) states that trustworthiness evidence includes development-stage evidence, delivery-stage evidence, and application-stage evidence.

Additionally, with the introduction of third-party components, their features cannot be assessed with traditional quality attributes. Emerging software quality concepts arise, such as configurability, customizability, reusability, scalability, availability, trackability, and compatibility to the quality requirements for software products produced by traditional software development processes (Challa et al. 2011; Chau 1994). These concepts emphasize the needs for third-party components to be flexible in design and application and easy to reuse, change, and extend. The object is not only to make the third-party components reusable but also to make them adapt components to end-users systems and requirements (Wang et al. 2019).

OSS is a frequently discussed topic in quality-related manuscripts. The advantage of OSS projects is that project data is public, so there are "many eyeballs" to help find and fix defects (Mohagheghi and Conradi 2007), thus ensuring more credible software quality. However, there are some warnings about the risk of having "too many cooks in the kitchen" (Androutsellis-Theotokis et al. 2011). We did not find clear evidence that the number of contributors correlates with software quality. However, research shows individuals may not access the software to test it directly. They prefer metrics such as the number of stars and users on GitHub as indicators of quality assessment (Li et al. 2021). In addition, the nature of OSS is that it is available to all end-users. Therefore, it is also susceptible to hackers, which opens a backdoor for the software and makes the system insecure. Furthermore, end-users do not necessarily suggest that all of them are concerned with the security of the software, as most of the people who use the software are engineers, not security experts.

Source Code Quality and Architecture Researchers calculate software quality as the ratio of the total number of defects identified in a certain period to the total number of code lines (Liu and Iyer 2007). Thus, it is not difficult to find that software quality, especially for OSS, depends on the quality source code (Qian et al. 2009) and its attributes, for instance, availability, understandability, completeness, conciseness, portability, consistency, maintainability, testability, usability, reliability, structuredness, and efficiency (Crowston, Annabi and Howison 2003; Jadhav and Sonar 2009). A frequently stated impact factor regarding source code is availability (Badampudi, Wohlin and Petersen 2016; Chau 1994; Crowston, Annabi and Howison 2003). Reasons abound for this, one of which may be that end-users or downstream engineers need to inspect or analyze source code to determine code quality and software quality.

Programming languages may not directly affect software trust. However, experts believed that projects written in more popular programming languages are more successful in market penetration and attracting human resources (Ghapanchi and Tavana 2015), indirectly giving end-users greater confidence in choosing their software. Several aspects need to be considered when selecting a programming language for a project: (1) the project itself, for instance, as a back-end server, it is not possible to consider a low-performance interpreted language like Python (Jadhav and Sonar 2009); (2) support of programming languages, for instance, whether in-house experts are proficient in the language (Jadhav and Sonar 2011); (3) dependencies and interfaces, for instance, whether they need to be purchased or used (Jadhav and Sonar 2011).

We try to collect a set of indicators for source code analysis and provide the result[3] in an open repository, which is also the frequently used field in various source code analysis tools. In forming this set of indicators, we considered, as much as possible, both the goal of the analysis (i.e., to be able to describe the quality aspects of the software process and the resulting product) and the availability of metrics in the data (Koch and Neumann 2008), assessing process and product evidence (Donohue, Dugan and Brown 2005).

The software architecture highly influences the selection of the software. It is reported that *"engineers will dismiss an unfamiliar framework and library if they cannot make it work within an hour"* (Haenni et al. 2013, 2014). Most of the time, end-users select a package or a library to implement a single function instead of an extensive system or application; for instance, for brownfield development, the existing software architecture of the project imposes constraints on the selection of the following software package or library, the selected package or library must fit into the existing technology stack of the project (Vargas et al. 2020). As software packages are hosted off-premises, they are challenging to integrate with other systems or software packages. Hence, end-users and software producers should pay additional attention to the architecture's integration, scalability, and reliability. They should ensure that software products can remain available within a given time window and maintain a reasonable response time even during peak periods (Godse and Mulik 2009).

Versions and Dependencies The distribution, installation, and updating of software packages or components also impact software trustworthiness, especially the integrity and reliability of the software. Distribution requires that the package or component is in the format provided by the software producer without any malformations or corruptions, and the installation and updates must also comply with the requirements set forth by the software producer (Catuogno, Galdi and Persiano 2017). However, dependencies complicate the software components' distribution. Dependencies connect various components of the system, thus forming a highly interconnected ecosystem (Hejderup, Deursen and Gousios 2018). Usually, the version that the package depends on is specific; therefore, a critical activity that must be carried out at each stage of the SDLC is ensuring the integrity and coherence of dependencies with deployment tasks, as failure in this process can result in disastrous consequences (Catuogno, Galdi and Persiano 2017).

Considerable security vulnerabilities are caused by aging libraries (Kula et al. 2015). However, it does not mean that a new version guarantees bug-free software. Not only that, some uncontrollable dependencies often pose higher risks. For instance, the left pad is a few-line software package with thousands of dependent packages. When its producer decided to delete it from npm, it *almost broke the internet*. Therefore, the requirements for a package or component are: (1) it has to be authentic; (2) unmodified; (3) fresh; (4) all required dependencies are adequately satisfied and ensure dependency, integrity, and consistency; (5) unauthorized installation is prohibited (Catuogno, Galdi and Persiano 2017).

Extrinsic Trust Factors

Extrinsic trust factors refer to the external environment characteristics that influence software selection. Although the majority of manuscripts discuss intrinsic

[3] https://data.mendeley.com/api/datasets/xn4jk93g4t/draft/files/b36ddc93-0f42-4227-b966-4c07d2a7e429?a=9c51b45f-c943-4a22-814f-5baf59fb2896

trust factors, end-users still prioritize some extrinsic trust factors in the software selection. This is because extrinsic trust factors have a wide range of data sources on publicly accessible portals compared to intrinsic trust factors that lack quantitative metrics or detailed data.

Structural Assurance In Section 4.1, we underlined that SA is believed to increase software trust. This is because software trustworthiness may increase when end-users perceive that they are using software in a secure, protected environment. SA includes licenses (e.g., GNU General Public License 2.0, or Apache License 2.0), contracts, regulations, or other safeguards (e.g., package promises and legal considerations) (Guo et al. 2014). Contracts define rights and obligations in a software project, including the functionality and goals of the software, what each stakeholder is expected to accomplish, and the solutions to anticipated risks or problems. License is the most important type of structural assurance, which has been stated in several manuscripts (Del Bianco et al. 2011; Jadhav and Sonar 2009; Vargas et al. 2020). It restricts the end-user's ability to use the software (Midha and Palvia 2012). For instance, the GPLv3 (General Public License) license requires developers to open their software, which is not a choice for commercial software (Vargas et al. 2020). In addition, different versions of the same software product may have different licenses, which end-users should be aware of during their software selection.

Documentation Documentation provides the first impression to end-users of a product's quality (Vargas et al. 2020). It includes the official project documentation, e.g., requirements documentation, architecture, technical and user manual (Sarrab and Rehman 2014); and broader forms of documentation, e.g., documentation translation, user comments, or discussion in the forums (Chau 1994). Crowston, Annabi and Howison (2003) argue that the organization, clarity, and maintenance of documentation are criteria that are often used to measure a software product. Our previous study seems to be consistent with this finding. We interviewed 12 participants, including software engineers, DevOps, architects, and other experts, and ten of them considered documentation crucial to package selection. Typically, they examine data structures, the functionality of features, and the package's compatibility with the project via documentation, one of them confirmed that packages requiring two days of reverse engineering due to unclear documentation would be eliminated (Jansen, Jansen and Hou 2021).

However, there is a problem with official and broader forms of documentation. In terms of the official documentation, not all types of documentation are available to the public. For example, to keep the trade secret, proprietary closed source software producers cannot share their project data and source code with the public (Alexy, Henkel and Wallin 2013). In terms of broader forms of documentation, although the documentation for open-source software or packages is free and open, much of the technical information and discussion occurs on mailing lists or forums, making it difficult to collect and maintain valid information, let alone version control of the documentation (Bangerth and Heister 2013).

Popularity Popularity is the most straightforward way to determine how widely used and liked the product is by the public. Popularity can be measured as the number of likes, downloads or dependencies of the software product (Vargas et al. 2020), or the number of subscribers or views of the information page (Crowston, Annabi and Howison 2003). Additionally, the number of dependencies reflects the

popularity of the software as a component or library. More dependencies mean more people are using it, and it is more stable (Vargas et al. 2020).

Reputation Reputation can be at the software level or human level. Human-level reputation is discussed in Section 4.2.3, the focus here is on the reputation of software. A good reputation represents a good rating by a specific size of users over a given period, and it represents a lower risk of choosing a software product (Duan et al. 2021). Therefore, with similar software quality, the end-users choose the software with a higher reputation. The most direct reputation indicator is the average software product rating (Alarcon et al. 2020). The reputation of a software product may include the number of releases or versions, product history, and open/closed issues (Alarcon et al. 2020), the programming language used, clarity of comments, number of bugs reported, tests performed, number of versions, and reviews from users who used the software or stakeholders who participated in the development or testing (Haefliger, Von Krogh and Spaeth 2008).

Cost Although it is widely accepted that cost is associated with software selection because solution selection frequently involves a trade-off between quality and cost (Limam and Boutaba 2010), there are still partial software end-users, such as some OSS adopters, who do not believe that cost is associated with software trust. This is most evident in the growing awareness of software, particularly OSS, across various industries. However, some specialized industries, such as banking, resist adopting inexpensive or even free software, such as OSS, despite the high cost of customizing and maintaining the software (Talib, Alsaafin and Medjden 2020). This finding is derived from a top 10 American bank case study. The main reason they are reluctant to use OSS is that they *"fear about who would maintain, fix, and otherwise support the systems"*. As a result, they adopted OSS in some non-core modules, such as the interfaces or customer self-service modules (Walli, Gynn and Von Rotz 2005), or moved some of their existing operating systems or databases to OSS operating systems or databases, to balance the costs associated with the high-priced core systems.

Top management from IT organizations, especially profit-oriented organizations, tend to consider the return on investment (ROI) as the primary financial cost factor in their software selection decision-making process (Del Bianco et al. 2011). ROI is the ratio of net benefits to costs. Mohagheghi and Conradi (2007) define the net benefits as savings, which are influenced by reuse costs and rework savings. Reuse cost includes the cost of developing reusable assets, integrating reusable assets, rework savings, and recovering creation costs. Rework savings measure the cost savings when adopting a trusted library or package, as well as savings in rework due to reduced system errors. However, the two factors are difficult to gather before the end-users select a software product. The possibility is that they are predicted or calculated after the software has been in use for a time. In the literature, costs usually refer to license cost, training cost, installation and implementation cost, maintenance cost, mitigation cost, and upgrading cost (Jadhav and Sonar 2009, 2011). Cho et al. (2019) introduce several cost factors as well, e.g., service costs and learning defense costs, in their model of trustworthiness of Computer-Based Systems. Service costs include defense costs, which refer to the cost of deploying various alternative solutions; decision costs, which refer to the administrative delays, complexity, and effort incurred in the decision process; and recovery costs, which refer to the costs associated with the recovery process to back to a normal operating system state. Recovery costs include intrusion response,

replacement, physical storage repair, and user training costs. Learning defense costs are the costs associated with implementing the system's defense mechanisms. In addition to the financial cost, the time cost, such as time to market and project completion time, and potential cost risks that can impact the cost also need to be considered (Mohagheghi and Conradi 2007; Vargas et al. 2020).

4.2.2 RQ2.2 - Trust Factors of Software Package Managers

This subsection focuses on addressing RQ2.2: *What trust factors do end-user organizations consider when selecting software package managers?*

A package manager is a software tool used to search, install, track, upgrade, and delete packages on the operating system. Based on the definition of the SECO, a package manager is also a SECO, with a set of actors interacting on this common platform. Perhaps it has been overlooked to some extent, at least so far, there has been little research concerning the trust relationships between the various hubs and actors in the package manager ecosystem, we only found four manuscripts (Catuogno, Galdi and Persiano 2017; Decan, Mens and Constantinou 2018; Duan et al. 2021; Hejderup, Deursen and Gousios 2018). The topics discussed are, respectively, how does the package manager ensure that the packages it manages are trustworthy, how does the package manager ensure that the packages are up-to-date, and how does the package manager detect vulnerabilities and notify end-users promptly and address the problems caused by the vulnerabilities. Therefore, the factors that contribute to trust in the package manager are limited, focusing on (1) dependency hell; (2) security vulnerability; (3) package prevention. They are all intrinsic trust factors related to the design of the package manager and its vulnerabilities.

Dependency Hell As previously stated, package installations are currently experiencing dependency hell, and end-users become trapped in a dead-end loop that state-of-the-art package managers cannot resolve. Catuogno, Galdi and Persiano (2017) argue that a trustworthy package manager should avoid the dependency hell by selecting the right package sources and versions, especially guaranteeing confidentiality, integrity, freshness, and authenticity of the packages and dependencies.

Security Vulnerability Security vulnerability includes two aspects, bugs and attacks on the metadata and the repository. As explained in Section 2, metadata includes the version of a package and a set of dependencies. Attackers may attempt to manipulate metadata to insert malicious code in dependencies to infect popular packages (Duan et al. 2021). In terms of the repository, if an online repository contains malicious code when downloading and upgrading packages, it will be imported into the end-user's computer through transitive dependencies (Hejderup, Deursen and Gousios 2018).

Prevention The protection mechanisms of package managers are discussed in two articles. Catuogno, Galdi and Persiano (2017) emphasize that multiple layers of protection should be provided to secure packages from their sources. For example, authentication of packages by digital signatures, verification of metadata freshness, and updating or downloading packages from multiple sources. Decan, Mens and Constantinou (2018) suggest that package managers should detect and deprecate vulnerable packages and warn end-users when packages with known vulnerabilities are installed or depended upon.

4.2.3 RQ2.3 - Trust Factors of Software Producing Organizations

This subsection focuses on addressing *RQ2.3. What trust factors do end-user organizations consider when selecting software producing organizations?*

In most cases, a product that is managed by an organization is more trustworthy because it gives end-users the confidence that the product is produced and managed by professionals (Hoxmeier 2000). However, it is often difficult to determine if an organization is a good self-promoter or a genuinely reputable organization. Next, we discuss the role of SPOs in software trust assessment.

Reputation Reputation is a publicly recognized judgment about the character or status of a person or thing (Immonen and Palviainen 2007). It is an asset for software producing organizations and gives them a competitive edge over others (Pollock and Williams 2007). Cai and Zhu (2016) argue that software quality has a positive relationship with an SPO's reputation, while Vargas et al. (2020) observe that an organization's reputation depends not only on the quality of the products it produces but also on other aspects, for instance, the services it provides (Jadhav and Sonar 2011), end-users feedback, and its rating values, positive and timely response, compliance with contracts, and length of experience (Capra et al. 2011; Challa et al. 2011; Chau 1994; Jadhav and Sonar 2009; Vargas et al. 2020). Length of experience, from an SPO's perspective, means that the SPO has experience in product manufacturing, project management, business operations, industry, and experienced in-house experts and engineers (Chau 1994). This extensive experience results in the SPO developing higher quality and trustworthy software. Especially when risks and uncertainties hit, they can deal with or avoid problems more effectively, increasing end-user confidence in their products and services.

Capability SPO's cability arises from different factors, including technology skills, the firm's strategy, maturity, and scale.

Technology skills are undoubtedly the capital of organizations to achieve success and are a core competency. Technology skills emphasize technical knowledge in software project management, including knowledge in programming and system development knowledge, R&D capability, training, and support. Trustworthy software cannot be developed without programming and system development knowledge, which is the physical guarantee of software production (Cho et al. 2019; Lai, Tong and Lai 2011; Sarrab and Rehman 2014). It means that the organization must have qualified members who are proficient in programming, system analysis, and design, project management, and other soft skills to deliver software on time, on quality, and on schedule, as well as effectively control and resolve all risks and issues in the SDLC (Amoroso et al. 1991). However, it is difficult to measure how many qualified members an organization has, and whether those members are qualified, one approach to measuring this would be to examine the software products and services offered by the organization. R&D capability shows whether an SPO can develop and maintain current projects and whether it has the potential and innovation to develop new products. It keeps the SPO ahead of the competition by continuously innovating and introducing new products and services or improving its existing offerings. End-users prioritize getting training and guidance from the services (Del Bianco et al. 2011; Jadhav and Sonar 2011; Sarrab and Rehman 2014; Vargas et al. 2020), which include: (1) training and documentation, e.g., user manual, tutorials, or troubleshooting guide; (2) maintenance and up-gradation, e.g., technical support and consultancy, communication with end-users, on-site

demo and free-trial version, rapid responsiveness, business skills (Jadhav and Sonar 2009, 2011), or management and operation (Hunter and Walli 2013); (3) social network-continued activity; (4) knowledge-sharing user-base (Wang et al. 2015b), e.g., discussion forums or signs of an active community. Especially for OSS, guaranteed, long-term technical support is an essential indicator for selecting an OSS software (Alarcon et al. 2020).

An SPO cannot produce a product in SECO without relying on the product of another SPO. Therefore, end-users' selection of an SPO should be accompanied by consideration of how the candidate SPO manages the SPOs on which it depends. For instance, how to select third-part components, manage third-party producers, or maintain the sourcing strategy. The strategy includes the SPO's management and vision, the SPO's culture, and the type of industry.

SPO's maturity indicates whether an SPO can manage the product well, which is essential to ensure product quality. It can be expressed through, for instance, content management systems, partnership models, rules, and regulations (Jansen 2014). Although there is no consensus on how to measure it, from the perspective of software end-users, formal procedures and standard security policies will help them evaluate and select trustworthy software products (Vargas et al. 2020).

SPO's capabilities are also affected by the number of contributors in the SPO (Bogart et al. 2016). For instance, how many people write code, report problems, and fix defects (Mockus, Fielding and Herbsleb 2002), as well as how effective these individuals are (Challa et al. 2011; Jansen 2014). Numerous studies have shown that the number of contributors positively impacts the success of a project (Crowston, Annabi and Howison 2003; Jansen 2014; Koch and Neumann 2008; Midha and Palvia 2012; Scacchi 2007; Sen, Singh and Borle 2012), especially for open source projects. By modeling the number of contributors and subscribers, the results suggest that more contributors to a project may increase the project activity and thus increase the number of subscribers. Similarly, increasing subscribers can affect the number of contributors by attracting new contributors and converting new contributors from the subscriber base (Sen, Singh and Borle 2012).

In our study, we found that SPOs are mentioned more frequently in OSS studies (Cruz, Wieland and Ziegler 2006; Liu and Iyer 2007; Midha and Palvia 2012; Sen, Singh and Borle 2012). Among them, it has been argued that SPO involvement and management in OSS development does not guarantee product quality and service (Capra et al. 2011). Because software engineers have to adhere to strict deadlines to meet SPO's goals and maintain leadership, at the expense of internal design quality.

4.2.4 RQ2.4 - Trust Factors of Engineers

This subsection answers *RQ2.4. What trust factors do end-user organizations consider when selecting software engineers?* Similar to the trust factors of SPOs, the trust factors of software engineers are also extrinsic. These trust factors, especially software engineers' competencies, and skills, contributing effort, satisfaction, and happiness, affect not only SPO trustworthiness but also software trustworthiness.
Competencies and Skills Competencies and skills refer to the knowledge and abilities that an engineer or a group of engineers should possess throughout the product's life cycle. They stem from good working knowledge, the experience of different operating systems, and the ability to learn from others (Sarrab and Rehman

2014). These skills and abilities, for example, the deployment and maintenance of software, the monitoring of news articles and messages in user forums and mailing lists regularly, and promptly responding to bug reports, are used to develop and integrate new and stable modules that meet end-user requirements (Norris 2004). Several researchers believe that they significantly impact the quality of the product they create (Mcknight et al. 2011; Norris 2004; Sarrab and Rehman 2014). In particular, engineer responsiveness is considered critical regardless of whether end-users trust the software or its output (Gefen and Keil 1996). Software engineers will likely suffer from distrust if they fail to respond to end-users promptly. While response times depend on the availability of engineers on the project and the priority of defects, in most cases, end-users are not concerned with them, but rather with defects being resolved quickly (Mockus, Fielding and Herbsleb 2002).

Contributing Effort Cai and Zhu (2016) believe that contributing effort helps shape an engineer's reputation. This is because engineers with low reputations may mean that they do not work hard enough, which leads to low contributions. Engineers with low contributions may lead to poor performance of the software products they produce.

Satisfaction and Happiness Researchers believe that engineers' satisfaction and happiness with working significantly affect the quality of the product of their products and the trust that end-users place in them (Capra et al. 2011; Crowston, Annabi and Howison 2003). Engineer satisfaction and happiness are highly subjective factors because each individual's satisfaction and happiness are unique. They are difficult to collect, analyze and quantify objectively.

Numerous research findings disagree on end-users importance in knowing who developed the software. This may imply that the end-users do not consider the software engineer to be a factor in their software selections. According to a set of interviews with 75 engineers that had been conducted in (Haenni et al. 2013, 2014), when the end-users select a component or library, they are not concerned with the engineers' identities but with the code's quality and level of maintenance, e.g., the engineers' responsibilities and their response time. Similarly, according to another study of an interview on package selection (Bogart et al. 2016), package selection depends on whether end-users trust the package maintainer. It is reported that one interviewee even deliberately sent bug reports to test whether the maintainer would respond in time. Nevertheless, we see some contrary views in certain manuscripts as well. For example, some end-users require that the names and numbers of engineers who develop and test the code must be public because they believe that reputable engineers help make code more reliable and reduce more defects (Madanmohan and De 2004). Both views imply that how engineers maintain the project is crucial to software selection.

5 Discussion

5.1 Impact on the Software Selection

The factors discussed in this study may positively or negatively affect the software selection outcome. No factor can have an absolute positive or negative impact. For instance, vulnerabilities and attacks appear to be detrimental to software trust, as their presence significantly impacts the quality and credibility of software and the

reputations of software producers (Cho et al. 2019; Vargas et al. 2020). With the advancement of attack techniques and the expansion of scale and scope, no software can now claim to be secure enough on its own. However, if the software can employ an effective defense to ensure a lower number of attacks over time, and if it is capable of employing more rapid means of resolving the situation following an attack, we can assert that although vulnerabilities and attacks undermine the software, the effect on selection and trust is positive (Gefen and Keil 1996). Similarly, factors such as the software's quality, reputation, or the skills of the software producer can have a positive or negative effect. It means that if a software product is of high quality and has a positive reputation, it will have a beneficial effect on the software trust, and vice versa. However, almost all reputation information is based on a web source. To project a trustworthy image, software producers are likely to falsify information or provide only one-sided information to achieve a higher reputation (Farooq et al. 2016). For instance, numerous proprietary product licenses contain clauses prohibiting public criticism of the product without the software producers' prior permission. Hence, a single-source reputation makes it difficult for end-users to obtain real feedback on the software product and thus assess its trustworthiness (Tavakolifard and Almeroth 2012).

The negative or positive impact is determined by how the end-user perceives those factors for various reasons. For instance, as we have discussed previously, several end-users choose low-cost software, whereas in several industries, such as banking, the cost is not a factor that prevents them from making a choice; rather, it represents a certain level of service reliability. Or, different end-users have different views on whether OSS is developed by SPOs, with some believing that SPOs represent sophisticated management and others believing that SPOs may be overly concerned with deadlines at the expense of product quality.

There is one situation in which the impact is neither positive nor negative: the propensity of software end-users. It affects the end-user to conduct the trust process professionally and rationally. Usually, the effect on decision-making should be beneficial. However, if the technology of the software evaluation exceeds, such as end-user experience, it may have no impact on the decision outcome (Moyano, Fernandez-Gago and Lopez 2016).

5.2 Validity Consideration

We considered the validity of the following points. First, our selection of manuscripts was inevitably subjective due to bias and understanding of the concepts, so we could not have included all relevant studies in this SLR. Second, we found that most studies focused on software quality, with relatively few studies on hubs, actors, and their relationships in the SECO. Perhaps because OSS provides public access to technical data while proprietaries' data is restricted, the manuscripts we extracted are more relevant to OSS-based discussions. Hence, the analysis of the results may not cover all types of SECOs. Additionally, a considerable number of studies are based on interviews or questionnaires, which may cause the results to be not generalized. Moreover, most models are not yet widely adopted in the open-source ecosystem, which means they still lack validation and application in real-world projects, and may be problematic in terms of generalizability. Finally, we categorized software trust as intrinsic and extrinsic trust factors with different

levels. Although this allows for a clarified and focused presentation of the results, we may have inevitably removed several factors that could not be categorized and overlooked specific scenarios of end-user trust.

5.3 Lessons and Research Challenges

Software trust is comprehensive, and its factors cover all actors in the SECO. In this subsection, we identify several challenges that make research more complicated. The first complicating factor is that end-users propensity to trust is subjective. Even when different end-users are given the same information, the trust they place will be different, and this trust can change over time. The judgment depends on the information they access. Therefore, assigning a fixed priority or weight to the factors is impossible. Second, it is difficult, if not impossible, to find a comprehensive set of models to measure software trust objectively and fairly. The measurement and interpretation of software trust must be dynamic. Third, the interpretation of trust factors should be multifaceted. It may be challenging to obtain or apply project data to determine trust in closed software, but these data can be used to calculate trust in OSS.

The digitization of society has provided us with a number of affordances, and the penetration of IT in it should be considered an unprecedented development in history. However, we are standing at the advent of a new period in history, where societal trust in IT is dropping at an alarming rate (Public Affairs Council 2021). We can say the hay-day of society's willingness to adopt new IT solutions and trust the older ones is over (Hayes, Poel and Steen 2020). With this problem statement in mind, we have set ourselves an ambitious goal to raise the accessibility, reliability, and use of trust data in the worldwide SECO (Hou, Farshidi and Jansen 2021). We envision a future where software can be rapidly and even automatically assessed relatively and equally, using a shared understanding of trust that is collaboratively created and maintained. However, several challenges must be tackled before creating such an accessible and common infrastructure. We define a set of research challenges for the future in the **bold** typeface.

The first challenge is that we are collecting and sharing software engineer data, but at the risk of revealing their identities and breaking their privacy (Poel 2020). This would directly contradict the goals of this project, so we need to find mechanisms for **protecting personal data within ethical and legal boundaries**, such as the General Data Protection Regulation (GDRP).

There is an inherent problem with sharing information with a community, as actors could misjudge or misuse the information. It must be noted that **observed vulnerabilities are not a sign of untrustworthy software** and that if these vulnerabilities are rapidly identified and eliminated, they actually might be a sign of trustworthy software. Furthermore, we need to ensure that **vulnerabilities can be shared openly within the community without falling into the "wrong" hands**.

We have performed two preliminary interview studies to assess how software engineers and end-user organizations would use trust data in their work. These interview studies showed that while trust facts may be irrefutable, the perception of trust is a moving target for different actors. One actor may be satisfied with using a bleeding-edge software package that often updates, while another wants

an old version of a patched but reliable package. We are currently exploring if **different categorizations of trust ratings are necessary**, such as "project evolution speed", "vulnerability fix time", and "team liveliness".

While we respect tools that centrally collect trust data, we believe that trust data should be part of a zero-trust community. We observe that it is **impossible to observe correct trust data as one major party objectively** and consensus mechanisms are necessary to observe that a software package version is objectively trustworthy.

The data collection process itself is challenging. Once we have identified the values (that is, trust facts) that are necessary to collect, we will have to tackle the challenge of the availability of these data. Data to judge the software trust can come from usage data, source code, build systems, test systems, etc. Thus, it will be challenging to **judge the trust of closed software**. Furthermore, it is hard, if not impossible, to **get data from closed reputation databases**, such as app store ratings. Finally, the **operationalization of the impact factors** is a major challenge in this work. We plan to perform a large-scale survey study with software engineers as an approach to this problem to identify a minimal collectible set of trust facts useful for software engineers.

5.4 Future Work

Our near-future works consist roughly of two parts.

First, we plan to **launch a software engineer survey based on the outcomes of this SLR**. The survey intends to determine a prioritized and categorized list of trust factors. Furthermore, we want to use the survey to gauge and create awareness amongst software engineers about the role of trust in their software engineering processes.

Secondly, using the list of prioritized trust factors, **we develop a tool for automatically gathering trust scores for different trust impact categories** (Hou, Farshidi and Jansen 2021). Software engineers must evaluate the tool to decide whether it effectively supports them in downloading and depending on a particular software package. We aim to store these trust scores in a distributed ledger to make trust common to all software engineers. The community that uses the ledger can contribute openly observable facts to the ledger to increase or decrease trust scores for software packages.

6 Conclusion

In this study, we examined the role of trust in SECO by starting with the selection and measurement of software products. Our goal is to devise a strategy for resolving the trust erosion issue raised in the introduction, specifically the harm caused to SECO's health by vulnerabilities and attacks. This study adopted a systematic literature review approach to review the trust in SECO of a selected 112 manuscripts. On this basis, we discussed the definition, types, and sources of software trust and propose a definition of software trust in the context of SECO. Additionally, we classified the impact factors on software trust from the standpoint of software and packages, software package managers, software producers, and software end-users.

We analyzed the existing literature for relevant impact factors and counted their frequencies. The most frequently discussed factors are software quality, source code, security, documentation, and structural assurance. Furthermore, based on these studies, we compiled a comprehensive table of software trust impact factors and identified as many practical metrics as possible for each factor to be measured.

Although most of the trust factors we collected are based on empirical research and only a few models or metrics are widely used in real-world projects, they reflect, to some extent, the concerns expressed by software engineers or end-users during the software selection or evaluation process.

According to the literature, software trust is the foundation for cooperation between hubs and actors in SECO. While software trust is critical for cooperation between actors, current research on software trust focuses exclusively on software and components, particularly software quality. Limited measures of trust assessment cannot be the final word to success, and it should be expanded to include other entities to analyze trust in the SECO in a multi-level, multi-perspective manner. Next, we will survey software engineers to ascertain their perceptions of software trust in the future, paving the way for designing a community-managed infrastructure that serves as a trust layer for SECO.

Acknowledgements We thank the SecureSECO team for their constructive feedback and help to create this article. In particular, we thank Donny Groeneveld, Venja Beck, and Floris Jansen for their diligence in performing their part of the literature review, i.e., filtering, double-checking, and processing the articles. Furthermore, we thank them for the lengthy and productive discussions that led to an inter-rater agreement between the SLR team members. Finally, we thank Hidde Reeskamp and Joost Gadellaa for their excellent comments on the early versions of this article.

Declarations

Funding and Conflicts of interests

This project received funding from the TruBlo project the third Open Call (subgrant agreement), under European Union's Horizon 2020 Research and Innovation program under the Grant Agreement Number 957228.
Author Slinger Jansen is an Associate Editor at the Empirical Software Engineering Journal.

Systematic Literature Review Articles

Alarcon, Gene M et al. (2020). 'Trust Perceptions of Metadata in Open-Source Software: The Role of Performance and Reputation'. In: *Systems* 8.3, p. 28.

Amoroso, Ed et al. (1991). 'Toward an approach to measuring software trust'. In: *Proceedings. 1991 IEEE Computer Society Symposium on Research in Security and Privacy*. IEEE Computer Society, pp. 198–198.

Androutsellis-Theotokis, Stephanos et al. (2011). 'Open source software: A survey from 10,000 feet'. In: *Foundations and Trends in Technology, Information and Operations Management* 4.3-4, pp. 187–347.

Badampudi, Deepika, Claes Wohlin and Kai Petersen (2016). 'Software component decision-making: In-house, *OSS*, *COTS* or outsourcing-*A* systematic literature review'. In: *Journal of Systems and Software* 121, pp. 105–124.

Bennett, Keith et al. (2000). 'Service-based software: the future for flexible software'. In: *Proceedings Seventh Asia-Pacific Software Engeering Conference. APSEC 2000*. IEEE, pp. 214–221.

Berander, Patrik et al. (2005). 'Software quality attributes and trade-offs'. In: *Blekinge Institute of Technology* 97.98, p. 19.

Bogart, Christopher et al. (2016). 'How to break an API: cost negotiation and community values in three software ecosystems'. In: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 109–120.

Boyes, HA et al. (2014). 'Trustworthy Software: lessons from goto fail'& Heartbleed bugs'. In: *9th IET International Conference on System Safety and Cyber Security*, pp. 1–7.

Bugiel, Sven, Lucas Vincenzo Davi and Steffen Schulz (2011). 'Scalable trust establishment with software reputation'. In: *Proceedings of the sixth ACM workshop on Scalable trusted computing*, pp. 15–24.

Cai, Yuanfeng and Dan Zhu (Aug. 2016). 'Reputation in an open source software community: Antecedents and impacts'. In: *Decision Support Systems* 91, pp. 103–112. DOI: 10.1016/j.dss.2016.08.004.

Capra, Eugenio et al. (2011). 'Firms' involvement in Open Source projects: A trade-off between software structural quality and popularity'. In: *Journal of Systems and Software* 84.1, pp. 144–161.

Catuogno, Luigi, Clemente Galdi and Giuseppe Persiano (2017). 'Secure dependency enforcement in package management systems'. In: *IEEE Transactions on Dependable and Secure Computing* 17.2, pp. 377–390.

Challa, Jagat Sesh et al. (2011). 'Integrated software quality evaluation: a fuzzy multi-criteria approach'. In: *Journal of Information Processing Systems* 7.3, pp. 473–518.

Chau, PYC (1994). 'Selection of packaged software in small businesses'. In: *European Journal of Information Systems* 3.4, pp. 292–302.

Cho, Jin-Hee, Kevin Chan and Sibel Adali (2015). 'A survey on trust modeling'. In: *ACM Computing Surveys (CSUR)* 48.2, pp. 1–40.

Cho, Jin-Hee et al. (2019). 'Stram: Measuring the trustworthiness of computer-based systems'. In: *ACM Computing Surveys (CSUR)* 51.6, pp. 1–47.

Crowston, Kevin, Hala Annabi and James Howison (2003). 'Defining open source software project success'. In: *ICIS 2003 Proceedings*, p. 28.

Cruz, David, Thomas Wieland and Alexander Ziegler (2006). 'Evaluation criteria for free/open source software products based on project analysis'. In: *Software Process: Improvement and Practice* 11.2, pp. 107–122.

Das, Tushar Kanti and Bing-Sheng Teng (2001). 'Trust, control, and risk in strategic alliances: An integrated framework'. In: *Organization studies* 22.2, pp. 251–283.

Decan, Alexandre, Tom Mens and Eleni Constantinou (2018). 'On the impact of security vulnerabilities in the npm package dependency network'. In: *Proceedings of the 15th International Conference on Mining Software Repositories*, pp. 181–191.

Del Bianco, Vieri et al. (2011). 'A survey on open source software trustworthiness'. In: *IEEE software* 28.5, pp. 67–75.

Donohue, Susan K, Joanne Bechta Dugan and Charles L Brown (2005). 'Is My Software" Good Enough" to Release?-*A* Probabilistic Assessment'. In: *29th annual IEEE/NASA Software engineering workshop*. IEEE, pp. 5–13.

Duan, Ruian et al. (2021). 'Towards Measuring Supply Chain Attacks on Package Managers for Interpreted Languages'. In: *Network and Distributed Systems Security (NDSS) Symposium 2021*.

Ellison, Robert, William Nichols and Carol Woody (2016). 'Measuring software assurance'. In: *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 2. IEEE, pp. 359–364.

Gefen, David and Mark Keil (1996). 'DEVELOPER RESPONSIVENESS AND PERCEIVED USEFULNESS.' In: *Academy of Management Proceedings*. Vol. 1996. Academy of Management Briarcliff Manor, NY 10510, pp. 313–317.

Ghapanchi, Amir Hossein and Madjid Tavana (2015). 'A longitudinal study of the impact of open source software project characteristics on positive outcomes'. In: *Information Systems Management* 32.4, pp. 285–298.

Godse, Manish and Shrikant Mulik (2009). 'An approach for selecting software-as-a-service (SaaS) product'. In: *2009 IEEE International Conference on Cloud Computing*. IEEE, pp. 155–158.

Goode, Sigi et al. (2015). 'Rethinking the *Role* of Security in Client Satisfaction with *Software*-as-a-*Service* (*SaaS*) Providers'. In: *Decision Support Systems* 70, pp. 73–85. DOI: 10.1016/j.dss.2014.12.005.

Grandison, Tyrone and Morris Sloman (2000). 'A survey of trust in internet applications'. In: *IEEE Communications Surveys & Tutorials* 3.4, pp. 2–16.

Grodzinsky, Frances S, Keith W Miller and Marty J Wolf (2011). 'Developing artificial agents worthy of trust:"Would you buy a used car from this artificial agent?"' In: *Ethics and information technology* 13.1, pp. 17–27.

Guo, Guibing et al. (2014). 'From ratings to trust: an empirical study of implicit trust in recommender systems'. In: *Proceedings of the 29th annual acm symposium on applied computing*, pp. 248–253.

Haefliger, Stefan, Georg Von Krogh and Sebastian Spaeth (2008). 'Code reuse in open source software'. In: *Management science* 54.1, pp. 180–193.

Haenni, Nicole et al. (2013). 'Categorizing developer information needs in software ecosystems'. In: *Proceedings of the 2013 international workshop on ecosystem architectures*, pp. 1–5.

— (2014). 'A quantitative analysis of developer information needs in software ecosystems'. In: *Proceedings of the 2014 European Conference on Software Architecture Workshops*, pp. 1–6.

He, JiuSong et al. (2009). 'Reference Model of Trustworthy Proof for Trusted Components'. In: *2009 Second International Conference on Future Information Technology and Management Engineering*. IEEE, pp. 136–139.

Heiskanen, Ari, Mike Newman and Merja Eklin (2008). 'Control, trust, power, and the dynamics of information system outsourcing relationships: A process study of contractual software development'. In: *The Journal of Strategic Information Systems* 17.4, pp. 268–286.

Hejderup, Joseph, Arie van Deursen and Georgios Gousios (2018). 'Software ecosystem call graph for dependency management'. In: *2018 IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results (ICSE-NIER)*. IEEE, pp. 101–104.

Hillebrand, Channel and Marijke Coetzee (2013). 'Towards reputation-as-a-service'. In: *2013 Information Security for South Africa*. IEEE, pp. 1–8.

Hong, Huang, Wang Chang-hui and Wang Ben (2011). 'Research on Management Scheme of Trusted Application Software'. In: *2011 International Conference on Network Computing and Information Security*. Vol. 1. IEEE, pp. 311–315.

Hoxmeier, John A (2000). 'Software preannouncements and their impact on customers' perceptions and vendor reputation'. In: *Journal of Management Information Systems* 17.1, pp. 115–139.

Hunter, Paula and Stephen Walli (2013). 'The Rise and Evolution of the Open Source Software Foundation'. In: *IFOSS L. Rev.* 5, p. 31.

Immonen, Anne and Marko Palviainen (2007). 'Trustworthiness evaluation and testing of open source components'. In: *Seventh International Conference on Quality Software (QSIC 2007)*. IEEE, pp. 316–321.

Jackson, Daniel (2009). 'A direct path to dependable software'. In: *Communications of the ACM* 52.4, pp. 78–88.

Jadhav, Anil S and Rajendra M Sonar (2009). 'Evaluating and selecting software packages: A review'. In: *Information and software technology* 51.3, pp. 555–563.

— (2011). 'Framework for evaluation and selection of the software packages: A hybrid knowledge based system approach'. In: *Journal of Systems and Software* 84.8, pp. 1394–1407.

Jansen, Slinger (2014). 'Measuring the health of open source software ecosystems: Beyond the scope of project health'. In: *Information and Software Technology* 56.11, pp. 1508–1519.

Koch, Stefan and Christian Neumann (2008). 'Exploring the Effects of Process Characteristics on Products Quality in Open Source Software Development'. In: *Journal of Database Management (JDM)* 19.2, pp. 31–57.

Kula, Raula Gaikovina et al. (2015). 'Trusting a library: A study of the latency to adopt the latest maven release'. In: *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, pp. 520–524.

Lai, Ivan KW, Viny WL Tong and Donny CF Lai (2011). 'Trust factors influencing the adoption of internet-based interorganizational systems'. In: *Electronic Commerce Research and Applications* 10.1, pp. 85–93.

Limam, Noura and Raouf Boutaba (2010). 'Assessing software service quality and trustworthiness at selection time'. In: *IEEE transactions on software engineering* 36.4, pp. 559–574.

Liu, Xiang and Bala Iyer (2007). 'Design architecture, developer networks and performance of open source software projects'. In: *ICIS 2007 Proceedings*, p. 90.

Manikas, Konstantinos and Klaus Marius Hansen (2013). 'Software ecosystems–A systematic literature review'. In: *Journal of Systems and Software* 86.5, pp. 1294–1306.

McKnight, D Harrison (2005). 'Trust in information technology'. In: *The Blackwell encyclopedia of management* 7, pp. 329–331.

Mcknight, D Harrison et al. (2011). 'Trust in a specific technology: An investigation of its components and measures'. In: *ACM Transactions on management information systems (TMIS)* 2.2, pp. 1–25.

Midha, Vishal and Prashant Palvia (2012). 'Factors affecting the success of Open Source Software'. In: *Journal of Systems and Software* 85.4, pp. 895–905.

Mockus, Audris, Roy T Fielding and James D Herbsleb (2002). 'Two case studies of open source software development: Apache and Mozilla'. In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 11.3, pp. 309–346.

Mohagheghi, Parastoo and Reidar Conradi (2007). 'Quality, productivity and economic benefits of software reuse: a review of industrial studies'. In: *Empirical Software Engineering* 12.5, pp. 471–516.

Moyano, Francisco, Carmen Fernandez-Gago and Javier Lopez (2016). 'A model-driven approach for engineering trust and reputation into software services'. In: *Journal of Network and Computer Applications* 69, pp. 134–151.

Norris, Jeffrey S (2004). 'Mission-critical development with open source software: Lessons learned'. In: *IEEE software* 21.1, pp. 42–49.

Pollock, Neil and Robin Williams (2007). 'Technology choice and its performance: Towards a sociology of software package procurement'. In: *Information and Organization* 17.3, pp. 131–161.

Qian, Hongbing et al. (2009). 'Quality Process-Oriented Software Credibility Measurement and Assessment'. In: *2009 International Conference on Computational Intelligence and Software Engineering*. IEEE, pp. 1–6.

Rosenberg, Linda, Ted Hammer and Jack Shaw (1998). 'Software metrics and reliability'. In: *9th international symposium on software reliability engineering*.

Roumani, Yaman, Joseph K Nwankpa and Yazan F Roumani (2017). 'Adopters' trust in enterprise open source vendors: An empirical examination'. In: *Journal of Systems and Software* 125, pp. 256–270.

Sarrab, Mohamed and Osama M Hussain Rehman (2014). 'Empirical study of open source software selection for adoption, based on software quality characteristics'. In: *Advances in Engineering Software* 69, pp. 1–11.

Scacchi, Walt (2007). 'Free/open source software development: Recent research results and methods'. In: *Advances in Computers* 69, pp. 243–295.

Schuur, Henk van der, Slinger Jansen and Sjaak Brinkkemper (2011). 'The power of propagation: on the role of software operation knowledge within software ecosystems'. In: *Proceedings of the International Conference on Management of Emergent Digital EcoSystems*, pp. 76–84.

Sen, Ravi, Siddhartha S Singh and Sharad Borle (2012). 'Open source software success: Measures and analysis'. In: *Decision Support Systems* 52.2, pp. 364–372.

Van Den Berk, Ivo, Slinger Jansen and Lútzen Luinenburg (2010). 'Software ecosystems: a software ecosystem strategy assessment model'. In: *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, pp. 127–134.

Vargas, Enrique Larios et al. (2020). 'Selecting third-party libraries: The practitioners' perspective'. In: *CoRR*.

Wang, Baohua et al. (2019). 'Updating model of software component trustworthiness based on users feedback'. In: *IEEE Access* 7, pp. 60199–60205.

Wang, Huaimin (2011). 'TRUSTIE: design of a trustworthy software production environment'. In: *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE, pp. 3–4.

Wang, Huaimin et al. (2015a). 'TRUSTIE: a software development platform for crowdsourcing'. In: *Crowdsourcing*. Springer, pp. 165–190.

Wang, Jing et al. (2015b). 'Comparative case studies of open source software peer review practices'. In: *Information and Software Technology* 67, pp. 1–12.

Yan, Zheng (2008). 'A comprehensive trust model for component software'. In: *Proceedings of the 4th international workshop on Security, privacy and trust in pervasive and ubiquitous computing*, pp. 1–6.

Zhu, Ming-Xun et al. (2012). 'A non-functional requirements tradeoff model in trustworthy software'. In: *Information Sciences* 191, pp. 61–75.

Other Sources (marked as blue in text)

Alexy, Oliver, Joachim Henkel and Martin W Wallin (2013). 'From closed to open: Job role changes, individual predispositions, and the adoption of commercial open source software development'. In: *Research Policy* 42.8, pp. 1325–1340.

Bangerth, Wolfgang and Timo Heister (2013). 'What makes computational open source software libraries successful?' In: *Computational Science & Discovery* 6.1, p. 015010.

Bauer, Paul C (2019). 'Conceptualizing trust and trustworthiness'. In: *Political Concepts Working Paper Series*.

Bauer, Veronika, Lars Heinemann and Florian Deissenboeck (2012). 'A structured approach to assess third-party library usage'. In: *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, pp. 483–492.

Becerra, Manuel, Randi Lunnan and Lars Huemer (2008). 'Trustworthiness, risk, and the transfer of tacit and explicit knowledge between alliance partners'. In: *Journal of Management Studies* 45.4, pp. 691–713.

Bourque, Pierre and eds R.E. Fairley (2014). *Guide to the software engineering body of knowledge*. Version 3.

Buscemi, Nina et al. (2006). 'Single data extraction generated more errors than double data extraction in systematic reviews'. In: *Journal of clinical epidemiology* 59.7, pp. 697–703.

Cappos, Justin et al. (2008a). 'A look in the mirror: Attacks on package managers'. In: *Proceedings of the 15th ACM conference on Computer and communications security*, pp. 565–574.

— (2008b). 'Package management security'. In: *University of Arizona Technical Report*, pp. 08–02.

Chandra, Yanto and Liang Shang (2019). *Qualitative research using R: A systematic approach*. Springer.

Cooper, Harris M, James L Lindsay and Erika A Patall (2008). 'Research synthesis and meta-analysis'. In: *Applied Methods*, p. 344.

Decan, Alexandre and Tom Mens (2019). 'What do package dependencies tell us about semantic versioning?' In: *IEEE Transactions on Software Engineering*.

Drucker, Aaron M, Patrick Fleming and An-Wen Chan (2016). 'Research techniques made simple: assessing risk of bias in systematic reviews'. In: *Journal of Investigative Dermatology* 136.11, e109–e114.

Durach, Christian, Joakim Kembro and Andreas Wieland (Oct. 2017). 'A New Paradigm for Systematic Literature Reviews in Supply Chain Management'. In: *Journal of Supply Chain Management* 53. DOI: 10.1111/jscm.12145.

Farooq, Umar et al. (2016). 'A multi source product reputation model'. In: *Computers in Industry* 83, pp. 55–67.

Farshidi, Siamak, Slinger Jansen and Mahdi Deldar (2021). 'A decision model for programming language ecosystem selection: Seven industry case studies'. In: *Information and Software Technology*, p. 106640.

Hanus, Michael (2018). 'Semantic versioning checking in a declarative package manager'. In: *Technical Communications of the 33rd International Conference on Logic Programming (ICLP 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Hayes, Paul, Ibo van de Poel and Marc Steen (2020). 'Algorithms and values in justice and security'. In: *AI & SOCIETY* 35, pp. 1–23. DOI: 10.1007/s00146-019-00932-9.

Heyns, Marita and Sebastiaan Rothmann (2015). 'Dimensionality of trust: An analysis of the relations between propensity, trustworthiness and trust'. In: *SA Journal of Industrial Psychology* 41.1, pp. 01–12.

Hou, Fang, Siamak Farshidi and Slinger Jansen (2021). 'TrustSECO: A Distributed Infrastructure for Providing Trust in the Software Ecosystem'. In: *International Conference on Advanced Information Systems Engineering*. Springer, pp. 121–133.

Jansen, Floris, Slinger Jansen and Fang Hou (2021). 'TrustSECO: An Interview Survey into Software Trust'. In: *arXiv:2101.06138*. URL: https://arxiv.org/pdf/2101.06138.pdf.

Jansen, Slinger, Michael A Cusumano and Sjaak Brinkkemper (2013). *Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry*. Edward Elgar.

Jansen, Slinger et al. (2012). 'Shades of gray: Opening up a software producing organization with the open software enterprise model'. In: *Journal of Systems and Software* 85.7, pp. 1495–1510.

Keele, Staffs et al. (2007). *Guidelines for performing systematic literature reviews in software engineering*. Tech. rep. Technical report, Ver. 2.3 EBSE Technical Report. EBSE.

Kitchenham, Barbara (2004). 'Procedures for performing systematic reviews'. In: *Keele, UK, Keele University* 33.2004, pp. 1–26.

Li, Xiaozhou et al. (2021). 'Exploring factors and measures to select open source software'. In: *arXiv preprint arXiv:2102.09977*.

Madanmohan, T.R. and Rahul De (Dec. 2004). 'Notice of Violation of *IEEE* Publication Principles Open source reuse in commercial firms'. In: *Software, IEEE* 21, pp. 62 –69. DOI: 10.1109/MS.2004.45.

Meline, Timothy (2006). 'Selecting studies for systemic review: Inclusion and exclusion criteria'. In: *Contemporary issues in communication science and disorders* 33.Spring, pp. 21–27.

Petersen, Kai, Sairam Vakkalanka and Ludwik Kuzniarz (2015). 'Guidelines for conducting systematic mapping studies in software engineering: An update'. In: *Information and software technology* 64, pp. 1–18.

Poel, Ibo van de (2020). 'Core Values and Value Conflicts in Cybersecurity: Beyond Privacy Versus Security'. In: *The Ethics of Cybersecurity*, p. 45. DOI: 10.1007/978-3-030-29053-5_3.

Public Affairs Council (2021). *2021 Public Affairs Pulse Survey Report*. URL: https://pac.org/wp-content/uploads/Pulse_2021_Report.pdf.

Sonatype (2021). *2021 state of the software supply chain*. URL: https://www.sonatype.com/hubfs/Q3\%202021-State\%20of\%20the\%20Software\

%20Supply\%20Chain-Report/SSSC-Report-2021_0913_PM_2.pdf?hsLang=en-us.

Talib, Manar Abu, Areej Alsaafin and Selma Manel Medjden (2020). 'Application of Quality in Use Model to Evaluate the User Experience of Online Banking Software'. In: *Journal of Cases on Information Technology (JCIT)* 22.2, pp. 34–51.

Tavakolifard, Mozhgan and Kevin C Almeroth (2012). 'A taxonomy to express open challenges in trust and reputation systems'. In: *Journal of Communications* 7.7, pp. 538–551.

Walli, Stephen, Dave Gynn and Bruno Von Rotz (2005). 'The growth of open source software in organizations'. In: *A report*.

Wright, Stephen (2010). 'Trust and trustworthiness'. In: *Philosophia* 38.3, pp. 615–627.

Xu, Lai and Sjaak Brinkkemper (2007). 'Concepts of product software'. In: *European Journal of Information Systems* 16.5, pp. 531–541.

Zerouali, Ahmed et al. (2018). 'An empirical analysis of technical lag in npm package dependencies'. In: *International Conference on Software Reuse*. Springer, pp. 95–110.