

Article

A New Way of Cataloging Software Engineering from Grounded Theory

Gustavo Navas ^{1,2,*} and Agustín Yagüe ³

¹ Escuela Técnica Superior de Ingenieros Informáticos ETSIINF, Universidad Politécnica de Madrid (UPM), Campus de Montegancedo, Boadilla del Monte, 28660 Madrid, Spain

² IDEIAGEOCA Research Group, Universidad Politécnica Salesiana (UPS), Morán Valverde S/N y Rumichaca, Quito 170702, Ecuador

³ Escuela Técnica Superior de Ingeniería de Sistemas Informáticos ETSISI, Universidad Politécnica de Madrid (UPM), Calle Alan Turing s/n, Ctra. de Valencia, Km. 7, 28031 Madrid, Spain; agustin.yague@upm.es

* Correspondence: ge.navas@alumnos.upm.es; Tel.: +593-987030574

Abstract

Since software development is in constant evolution, it presents new facets and emerging topics. For example, the integration of qualitative data analysis through Grounded Theory establishes a new approach to categorizing this discipline, allowing it to align with these new contributions. The present work applies an original Glaserian Systematic Mapping Study (GSMS) to explore new ways of categorizing software development using Grounded Theory (GT) and GT elements. The study provides insights from the perspective of human beings, including their trust and doubts, and their attitudes towards work teams. The categorization covers many aspects of software development. One of these relates to Agile development, which has been defined in two pairs: Agile/Non-Agile and Agile/Plan-driven. Although this may seem obvious, this categorization had not been defined in this way before. Data, in all its diversity, accompanies the software development process throughout its entire lifecycle. Another finding of the present work is the concept of bridges, which correspond to the diverse interrelationships within software development. Grounded Theory in the context of software development has enabled the creation of various types of bridges. These bridges could be established between individuals within the same development team, between different areas of software development, or between developers and their tools and artifacts, among others. These findings can be highly diverse and can help software engineers unlock their potential and explore various options in software development.

Keywords: Glaserian Systematic Mapping Study; Grounded Theory; software development

Academic Editor: Christos Bouras

Received: 2 February 2026

Revised: 19 February 2026

Accepted: 23 February 2026

Published: 3 March 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

The incorporation of Grounded Theory (GT) into software engineering is important for several reasons. First, acknowledging the human element within software development processes addresses a significant issue in this field. GT has its roots in the Social Sciences [1] and therefore emphasizes understanding human behavior and the environment in which it operates [2,3]. This approach is vital for both individual and team performance in software engineering projects, influencing project costs and outcomes.

GT offers contributions to software engineering by providing methodologies that deepen qualitative analysis and are based on data from the Grounded Theory [1,4,5]. Unlike traditional approaches that overlook the complexities of human interaction and context [6], GT provides a comprehensive understanding of these factors, enabling more informed decision-making in software development.

Also, data saturation in GT ensures a systematic analysis, resulting in the emergence of robust theories that can improve software engineering practices [4,5], in both Glaserian (GGT) and Straussian (SGT) GT variants [7].

In the direct relationship between software engineering and human factors, GT offers additional techniques and tools that enrich this discipline. By providing a framework for categorizing software engineering phenomena, GT enables the generation of new knowledge in the form of theories, development innovation, and advancement in the field.

For fully understanding the integration of GT into software engineering, it is necessary to examine the previous efforts made in this direction. As is well known, GT plays a dual role as a qualitative research methodology and as a data analysis tool, due to its ability to extract knowledge from various sources. However, existing classifications of software development, such as those found in SWEBOK [8], often fail to incorporate the qualitative aspects emphasized by GT and the complex nature of software development processes.

While SWEBOK provides a rigorous and widely adopted technical taxonomy, it primarily organizes knowledge areas according to functional domains. It does not explicitly capture cross-cutting sociotechnical relationships, emergent human-centered dynamics, or relational patterns between development paradigms. GSMS does not seek to replace SWEBOK, but rather to complement it by introducing an analytical layer focused on emergent relational structures.

It is important to acknowledge that established frameworks, such as SWEBOK, are deeply embedded in academic curricula, certification systems, and industry standards. GSMS is not proposed as a substitute, but as an analytical complement that can coexist with standardized taxonomies. Its adoption would require institutional validation, methodological training, and integration with existing knowledge management systems.

Building on previous research that highlights the potential of GT in software development, despite its complexities and variants, this work seeks to investigate deeper into the integration of GT with other methodologies and techniques. Earlier studies have emphasized the importance of combining GT with approaches such as Systematic Mapping or empirical studies to address the challenges inherent in software development [9,10].

The research methodology employed, the Glaserian Systematic Mapping Study (GSMS), represents an innovative approach that merges Systematic Mapping with Grounded Theory principles [9]. By integrating GT's data analysis tools into the systematic Mapping process, GSMS enriches the analysis, enables the formulation of new research questions, and facilitates iterative development guided by predefined saturation equations [9,10]. Reaching saturation is a common GT terminology and has been formalized through three equations [9]. This methodology proceeds from traditional Systematic Mapping approaches by offering more profound insights and more specific answers to complex software engineering questions.

As a sociotechnical endeavor, software development relies heavily on effective communication among individuals. This techno-social challenge underscores the intersection of technical objectives with sociological perspectives. Recognizing the diverse range of data in software development, it becomes imperative to incorporate social science perspectives, particularly to understand the various stakeholders. In the GT methodology, every aspect is treated as data, underscoring the need for a holistic approach to analyzing software engineering phenomena.

The present research builds upon the groundwork laid by the Glaserian Systematic Mapping Study (GSMS) methodology, further advancing the integration of Grounded Theory into the categorization of software engineering products and artifacts. GSMS, characterized by its flexibility in analysis and result generalization, paves the way for proposing new theories and integrating novel forms of information, particularly those rooted in sociotechnical aspects.

One of the central theoretical outcomes of this study is the Bridge concept. A bridge is an emergent relation that links distinct categories while preserving their analytical identity. Bridges serve as analytical constructs that explain how diverse elements of software development interact within sociotechnical systems.

This paper is organized as follows. Section 2: Methodology delves into its previous studies within the qualitative and social analysis framework. Section 3 presents a study of Software Development using the GSMS methodology. Section 4 is the Discussion, and in Section 5 conclusions are drawn based on the final iterations.

2. Research Methodology

2.1. Overview of the GSMS Methodology

The Glaserian Systematic Mapping Study (GSMS) methodology was utilized in the present study. GSMS is a qualitative research methodology defined and introduced in the work [9], where its theoretical foundations and complete procedural details are described in depth. The methodology emerges from the integration of the Systematic Mapping Study (SMS) and Glaser's Grounded Theory (GGT), resulting in an approach that enables both systematic evidence Mapping and theory generation.

GSMS maps existing research while simultaneously developing emerging theories grounded in data, providing a deeper and more contextualized understanding of the phenomena under study [9]. A characteristic of GSMS is its iterative and adaptive nature, which allows for the refinement and creation of research questions and analytical strategies as concepts emerge throughout the analysis process [9,10]. This flexibility enables GSMS to capture the complexity and sociotechnical characteristics of software engineering phenomena.

The GSMS methodology [9] is instrumental in generating theories derived from the literature review efforts aimed at achieving saturation [9,11]. This section provides an overview of the three phases of GSMS: data collection, Data Analysis, and Comparison with the literature. Additionally, the termination of iterations is guided by the formal definition of saturation conditions within the GSMS methodology, which operationalizes theoretical saturation through three conditional statements (mathematical formulas) expressed as set-based relations between iterations, known as saturation criteria [9]. These conditions are formally defined as follows.

Condition 1—no emergence of new research questions:

$$Q_i = Q_{i+1} \quad (1)$$

where Q_i is the set of research question of iteration i . This condition indicates that iteration $i + 1$ does not generate additional research questions beyond those previously defined.

Condition 2—all research questions have at least one answer:

$$\forall RQ_j \in Q_i, \exists AQ_k \quad (2)$$

For every research question RQ_j that belongs to the set of research questions Q_i in iteration i , there exists at least one corresponding Answer Question AQ_k . This ensures that each research question has at least one associated answer.

Condition 3—no emergence of new answers:

$$AQ_i = AQ_{i+1} \quad (3)$$

This condition indicates stabilization of theoretical outcomes. When all three conditions are satisfied, theoretical saturation is reached.

The data collection phase resembles traditional SMS data-gathering processes, encompassing steps such as research question definition, search conducting, and paper screening, to collect pertinent information on the topic of interest [9,10].

Data analysis is conducted iteratively, constantly comparing collected data to construct theories until saturation is reached. It entails a structured approach akin to SMS Keywording, Mapping, and Synthesis, augmented by Grounded Theory processes like open, selective, and theoretical coding [9,10].

Within GSMS Data Analysis, the Glaserian Grounded Theory (GGT) stages—open, selective, and theoretical coding—yield specific outcomes: concepts, categories, and propositions, respectively [1]. These stages are integrated into the SMS Keywording, Mapping, and Synthesis steps within GSMS. Data collection yields concepts, which evolve into categories, and ultimately, theoretical statements manifest as propositions [9]. This paper adopts the presentation style proposed by Navas and Yagüe [9,10] to maintain clarity, where codes associated with concepts, categories, and propositions are denoted in braces, for example, {Grounded Theory} {Software development} [10].

The GSMS data analysis phase is conducted iteratively, with each successive iteration building upon the outcomes of the preceding one. This study consisted of five iterations, designated as iteration 0 through iteration 4. The iterations need the repeated execution of steps, such as Keywording, Mapping, and Synthesis, creating iterative feedback loops within the process. The zeroth and first iterations, together with their findings, were reported in a previous paper [10]. These initial iterations focused on establishing the foundational analytical framework by examining Grounded Theory, resulting in the identification of the Grounded Theory elements (GT_e) [10].

The present article builds upon that foundation and focuses on the subsequent stages of the GSMS analysis. While iterations 0 and 1 are centered on theory-oriented outcomes related to Grounded Theory itself, iterations 2 through 4 shift the analytical focus toward the domain of software engineering.

The comparison with the literature [1] phase involves collecting the theories from data analysis and juxtaposing them with the established literature. SMS additionally incorporates Rigor and relevance assessment in this phase [9,10].

2.2. GSMS Data Collection Phase (S1): Review

The present study adopts the GSMS data collection phase (S1) previously defined and executed in earlier work [9,10]. The data collection protocol is summarized here for comprehensiveness; all labels like S1, P1, etc., were defined in previous papers.

Goal: The GSMS data collection phase (S1) follows the SMS steps of (P1) research question definition, (P2) conduct search, and (P3) screening of papers [9].

Sources. In accordance with GSMS guidelines, the initial sources were selected from established scientific digital libraries, including Web of Science (WoS), IEEE Xplore, Scopus, and the ACM Digital Library [9,10].

Time window and execution date. The searches covered all years available up to 2023, considering indexed scientific articles from the selected libraries.

Search string definition. GSMS recommends balancing precision and generality when defining search expressions. Preliminary trials using complex multi-term expressions retrieved a limited number of relevant publications. Consequently, a simplified two-term expression (“Software” AND “Grounded Theory”) was adopted [9,10].

Deduplication and Screening.

The screening process initially identified 1750 records: 700 from Web of Science (WoS), 348 from IEEE Xplore, 418 from Scopus, and 284 from the ACM Digital Library [9,10]. The records were merged and screened using explicit inclusion and exclusion criteria. The inclusion criteria comprised: (i) peer-reviewed publications and (ii) primary studies.

The exclusion criteria were applied sequentially. The values reported below correspond to the number of articles remaining after applying each filtering step:

- (i) After removing duplicate records, 657 articles from WoS, 211 from IEEE Xplore, 154 from Scopus, and 238 from ACM Digital Library remained.
- (ii) After excluding non-English publications, 635 articles from WoS, 211 from IEEE Xplore, 154 from Scopus, and 238 from ACM remained.
- (iii) After excluding studies in which the term “Grounded” did not appear in the title or abstract, 579 articles from WoS, 192 from IEEE Xplore, 154 from Scopus, and 52 from ACM remained.
- (iv) After excluding studies that did not address Software Development as a primary topic, 365 articles from WoS, 192 from IEEE Xplore, 154 from Scopus, and 16 from ACM remained.
- (v) After excluding studies in which Grounded Theory (GT) was not a central topic, 127 articles from WoS, 67 from IEEE Xplore, 17 from Scopus, and 16 from ACM remained.
- (vi) Finally, after excluding studies in which neither Software Development nor Grounded Theory constituted the core focus of the investigation, 53 articles from WoS, 6 from IEEE Xplore, 2 from Scopus, and 9 from ACM Digital Library remained [9,10].

Snowballing: Following the initial filtering process, backward and forward snowballing were conducted to identify additional relevant studies, as recommended in GSMS. During the validated corpus construction, snowballing removed three review-type articles and added three new primary studies, resulting in a final corpus of 70 papers [9,10].

Final corpus: The final dataset comprises 70 primary studies, which constitute the corpus analyzed in the present paper [10].

2.3. GSMS Data Analysis Phase (S2): First Iteration Review

Iteration zero established the initial analytical conditions through a pre-classification schema, which provided a structured starting point for subsequent analysis.

The pre-classification schema consisted of two reference lists. The Grounded Theory list included its two main variants: Glaserian and Straussian. The Software Development list was derived from the software engineering Body of Knowledge (SWEBOK) and included the following knowledge areas: (1) software requirements, (2) software design, (3) software construction, (4) software testing, (5) software maintenance, (6) software configuration management, (7) software engineering process, (8) software engineering models and methods, (9) software quality, and (10) software engineering management [8–10].

This schema served as an initial analytical scaffold, enabling systematic comparison between GT approaches and software development domains throughout the iterative analysis.

The iteration zero. The initial conditions for GT and software engineering were established through the [pre-classification schema], which consisted of two lists: The GT list was formed with its two variants: Glaserian and Straussian. The Software Development list, obtained from SWEBOK [8], includes: 1. software requirements, 2. software design, 3. software construction, 4. software testing, 5. software maintenance, 6. software configuration management, 7. software engineering process, 8. software engineering models and methods, 9. software quality, and 10. software engineering management [9,10].

The first iteration focused on evaluating the impact of GT on software development and studying GT from the perspective of software engineering, with the fundamental finding being the Grounded Theory elements (GTe) [10].

In the second and subsequent iterations, software development is analyzed from the perspective of Grounded Theory and the findings of previous iterations. The second iteration is based on the GT and software development lists obtained from the {pre-classification schema}.

The findings of the first iteration, corresponding to GTe, are introduced in the third iteration. This iteration resulted in novel findings, which will be described in Section 3.2.

In the fourth iteration, it is verified that saturation has been reached by meeting the three equations proposed by the GSMS.

This section has been documented in a previous paper [10]. The present article develops the remaining iterations starting from the second.

2.4. Outcomes of the First Iteration

This subsection outlines the key outcomes of the first iteration as follows [10]:

1. {Pre-classification schema}
2. Six research questions

From the data collection phase:

1. RQ1: Where is Grounded Theory (GT) appropriate within the study of Software Development?
2. RQ2: Is GT applied correctly in the processes and tasks of Software Development?
3. RQ3: Is GT useful for software engineers in the industry?

From the iteration 0 phase:

4. RQ4: Is there a way to categorize documents within GT in Software Development to increase the percentage of cataloged papers?
 - a. Answer to RQ4: {AQ4₁: GTe}
5. RQ5: Are there other ways to categorize software beyond the pre-classification schema?
6. RQ6: How are GT elements associated with software development?

The concept of GT elements (GTe) emerged as a significant finding, particularly in response to RQ4, designated as AQ4₁: {GTe} [10]. Six elements form the unifying concept in GT studies within Software Development, referred to as {GT elements (GTe)}. These elements are essential to understanding the application of GT in this field. In summary, GT encompasses the following components: [9,10].

- {Data source}: The gathering of data through interviews, questionnaires, and observations.
- {Data analysis}: It involves various GT stages, such as open coding, selective coding, and theoretical coding, contributing to a comprehensive data analysis process [9].
- {Constant comparison}: It is utilized throughout the GT process to ensure consistency and depth in data analysis.
- {Basic Coding}: It includes concepts, indicators, patterns, categories, or initial properties forming the foundation of the GT structure.
- {Master Core category}: It represents the final category obtained after completing the GT process, encompassing propositions, recommendations, strategies, contingencies, consequences, changes, and more.
- {Emerging theory}: It arises from the core category and is articulated in two primary forms:
 - {A GT of ...}: It is the central theoretical statement addressing the article's subject.
 - {Use GT to ...}: It is the main theoretical statement guiding practical applications.

These elements are integral to effectively applying GT within Software Development. They provide a structured approach to analyzing and categorizing data.

3. Integrating GT in Software Engineering

Building upon two prior publications, this section explores the application of Grounded Theory (GT) to software engineering. The first publication identified the need to enhance systematic document reviews by merging Glaserian Grounded Theory (GGT) with Systematic Mapping Study (SMS), resulting in the Glaserian Systematic Mapping Study (GSMS) [9]. The second proposed a new classification schema for GT using GT elements (GTe) [10]. In the context of software engineering, the SWEBOK classification was utilized as a foundation, acknowledging that existing classification methods fail to represent a significant portion of articles. This work investigates whether GT, integrated with GSMS in its second iteration, can address this limitation by enabling a more comprehensive classification of software engineering.

This study treats articles selected for GSMS as transcripts for the GT application to generate theory. Traditionally rooted in analyzing diverse sources such as semi-structured interview transcripts, GT is extended here to scientific articles. Using GSMS phases—Keywording, Mapping, and Synthesis—enhanced with GT stages of open, selective, and theoretical coding, we assess the compatibility of this approach within software engineering. Specifically, open coding is initiated during Keywording, leveraging the Atlas.ti Version 25.0.2. software to automate the tagging and classification of word frequencies across selected scientific articles. To avoid confusion, word-frequency output is not treated as conceptualization. Instead, the lexical lists are used as a sensitizing and sampling aid to locate candidate meaning units (sentences/paragraphs) in the corpus. Open coding is then performed on those context-anchored excerpts through constant comparison and memoing, producing concepts that are subsequently integrated into categories and propositions (selective and theoretical coding).

The iterative process begins with open-coding Keywording, in which keywords are identified and categorized based on frequency and relevance. This systematic tagging and classification pave the way for further analysis during the Mapping and Synthesis phases. The integration of GT's analytical stages ensures a comprehensive examination of the data, allowing for the emergence of robust classifications and theoretical propositions. By incorporating these methodologies, we aim to establish a novel framework for understanding and categorizing software engineering phenomena.

3.1. Second Iteration: Integration of Grounded Theory (GT) in Software Engineering

A classification approach has been introduced, involving the analysis of interviews conducted on the articles, focusing on the most frequently recurring words.

According to the GSMS methodology, the data analysis involves continuously comparing data and abstractions through Keywording, Mapping, and Synthesis. This process utilizes open, selective, and theoretical coding to associate concepts, categories, and propositions. The goal of GSMS Keywording is to establish a classification schema based on existing knowledge from the literature. The complete workflow for the second iteration is shown in Figure 1.

The listed words made it easy to identify various concepts. Many of them were linked to the two lists of the pre-classification schema. Other concepts that could not be cataloged within the pre-classification schema were identified.

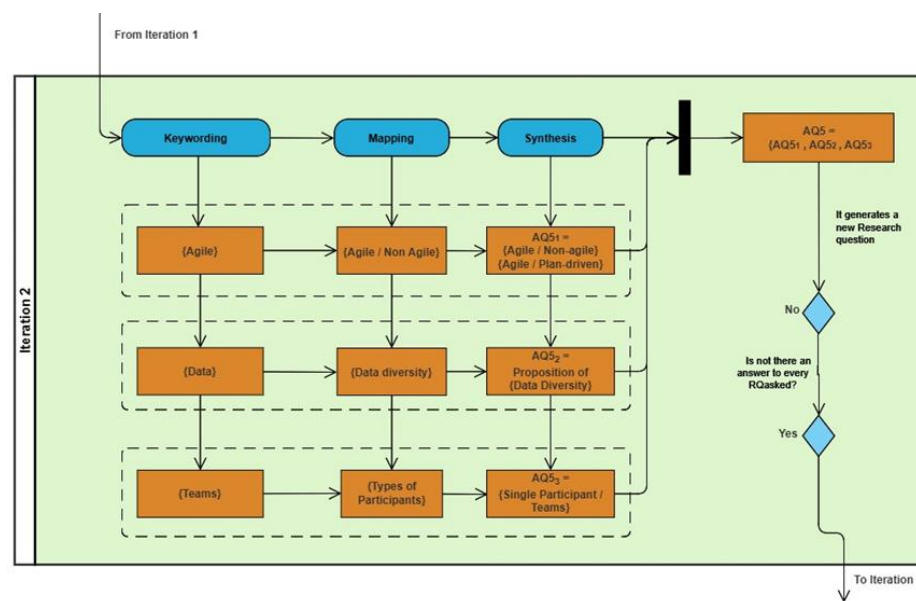


Figure 1. Activity diagram for the second iteration. showing in blue the steps of keywording, mapping, and synthesis; in brown the findings from each step; and the verification of saturation conditions for this iteration.

3.1.1. Keywording

The Keywording process was performed using the ATLAS.ti software. During the open coding phase, a comprehensive list of all lexical terms was extracted from the full-text content of the selected articles. These terms were sorted in descending order based on their frequency of occurrence, excluding common stopwords such as “a,” “and,” “of,” “to,” “is,” among others, which hold no analytical relevance. The token “The” is kept as a pipeline control item for quality assurance. It is not subject to filtering or analysis, and its inclusion is solely for validation and transparency.

The initial selection criterion identified words appearing more than once per article, yielding 4307 candidate terms. The Pareto Principle (the 80/20 rule) was applied to focus on the most representative vocabulary, reducing the dataset to the top 20% most frequent words (862 terms). To further refine this list using objective criteria derived from the data itself, we calculated two descriptive statistics based specifically on this subset of 862 high-frequency terms. First, we determined the mean article prevalence (i.e., the average number of articles in which these terms appeared), which was 35.1 papers. Second, we calculated the mean occurrence rate (i.e., the average frequency of these terms per article), which was 5.7 mentions. These mean values, inherent to our dataset, were then used as statistical thresholds to select only those terms performing above the average of this already highly relevant group. Consequently, the final set comprises words appearing in more than 35.1 articles and with a frequency greater than 5.7 mentions per article. This data-driven approach, which avoids arbitrary cutoffs by deriving thresholds from the corpus itself, yielded a robust set of 92 words. These keywords are presented in Table 1, which details (1) the total frequency of each term across the corpus, (2) the number of articles in which the term appears, (3) the average frequency per article, and (4) the assigned classification for each term. The classification was conducted by associating each term with one of two predefined schemes: terms relevant to software engineering were labeled P-C Swebok, following the Software Engineering Body of Knowledge (SWEBOK); terms related to Grounded Theory methodology were labeled P-C GT.

Additional terms that could not be categorized under these schemes were analyzed using open coding. Open coding led to the identification of three high-relevance

conceptual clusters: Agile, Data, and Team. These emerged as the most prominent concepts within the corpus and are positioned at the top of Table 1. Each group includes associated keywords that share a contextual or semantic affinity, which served as the basis for subsequent selective and theoretical coding phases in the study.

Table 1. Frequency, distribution, and classification of keywords in the analyzed articles.

WORDS	Total Count	Qty Papers	Frequency	Classification
The	33,125	-	-	CONTROL
Software	5154	70	73.6	P-C Swebok
Development	3148	70	45.0	P-C Swebok
Process	2584	70	36.9	P-C Swebok
Research	2175	70	31.1	P-C Swebok
Theory	2126	70	30.4	P-C GT
Data	1872	70	26.7	Data
Grounded	1345	70	19.2	P-C GT
Analysis	1187	70	17.0	P-C Swebok
Information	1220	69	17.7	P-C Swebok
Systems	885	69	12.8	P-C Swebok
Study	1806	68	26.6	P-C GT
Engineering	1206	68	17.7	P-C Swebok
Qualitative	546	68	8.0	P-C GT
Ieee	528	68	7.8	P-C Swebok
Technology	648	65	10.0	P-C Swebok
Context	485	65	7.5	P-C Swebok
Tools	640	64	10.0	P-C Swebok
Interviews	586	64	9.2	P-C GT
Knowledge	1102	63	17.5	P-C GT
Methods	1043	63	16.6	P-C Swebok
management	862	63	13.7	P-C Swebok
Projects	848	63	13.5	P-C Swebok
Coding	757	63	12.0	P-C Swebok
System	736	63	11.7	P-C Swebok
Processes	576	63	9.1	P-C Swebok
Project	1575	62	25.4	P-C Swebok
Quality	864	62	13.9	P-C GT
Method	763	62	12.3	P-C Swebok
Practice	484	62	7.8	P-C Swebok
Group	391	62	6.3	Teams
Categories	858	61	14.1	P-C GT
Model	654	61	10.7	P-C Swebok
Concepts	546	61	9.0	P-C GT
Questions	489	60	8.2	P-C GT
Empirical	431	60	7.2	P-C GT
Team	2394	59	40.6	Teams
Code	559	58	9.6	P-C GT
Role	512	58	8.8	Teams
Technical	491	58	8.5	P-C Swebok
Tasks	358	58	6.2	P-C Swebok
Product	1031	57	18.1	P-C Swebok
Pp	946	57	16.6	Agile

People	735	57	12.9	Teams
Communication	505	57	8.9	P-C GT
User	471	57	8.3	Teams
Literature	328	57	5.8	P-C GT
Being	326	57	5.7	P-C GT
Human	326	57	5.7	P-C GT
Requirements	1045	55	19.0	P-C Swebok
Participants	1001	55	18.2	Teams
Practices	692	55	12.6	P-C Swebok
programming	428	55	7.8	P-C Swebok
Theoretical	353	54	6.5	P-C GT
Developers	917	53	17.3	P-C Swebok
Category	528	53	10.0	P-C GT
Tool	402	53	7.6	P-C Swebok
Framework	508	52	9.8	P-C Swebok
Application	400	52	7.7	P-C Swebok
Researchers	346	52	6.7	P-C Swebok
Roles	303	52	5.8	Teams
Teams	1613	51	31.6	Teams
Business	725	51	14.2	P-C Swebok
Organization	482	51	9.5	P-C Swebok
Interview	457	51	9.0	P-C GT
Core	313	51	6.1	P-C GT
Source	367	50	7.3	P-C Swebok
Task	358	50	7.2	P-C Swebok
Proceedings	295	50	5.9	P-C Swebok
Companies	723	49	14.8	P-C Swebok
Organizations	481	49	9.8	P-C Swebok
Individual	280	49	5.7	Teams
Company	780	47	16.6	P-C Swebok
Test	632	47	13.4	P-C Swebok
Distributed	382	47	8.1	P-C Swebok
Models	297	47	6.3	P-C Swebok
Members	755	46	16.4	Teams
Developer	398	46	8.7	P-C Swebok
Glaser	330	45	7.3	P-C GT
Activity	266	45	5.9	Agile
Documentation	366	44	8.3	Teams
Organizational	315	44	7.2	P-C Swebok
Domain	288	44	6.5	P-C Swebok
Practitioners	264	44	6.0	Teams
Agile	2885	43	67.1	Agile
Challenges	248	43	5.8	
Users	378	42	9.0	Teams
Customer	596	41	14.5	Teams
Customers	344	40	8.6	Teams
Testing	949	39	24.3	P-C Swebok
Manager	357	39	9.2	P-C Swebok
Individuals	231	39	5.9	Teams

Selective coding grouped the terms into five main categories. Two of these categories correspond to the two lists defined in the pre-classification schema, shown on the left side of Figure 2: the Software Development list and the Grounded Theory list. The remaining three categories include terms that did not fit into either list and were therefore consolidated into three additional concepts: {Agile}, {Data}, and {Team}; shown on the right side of Figure 2.

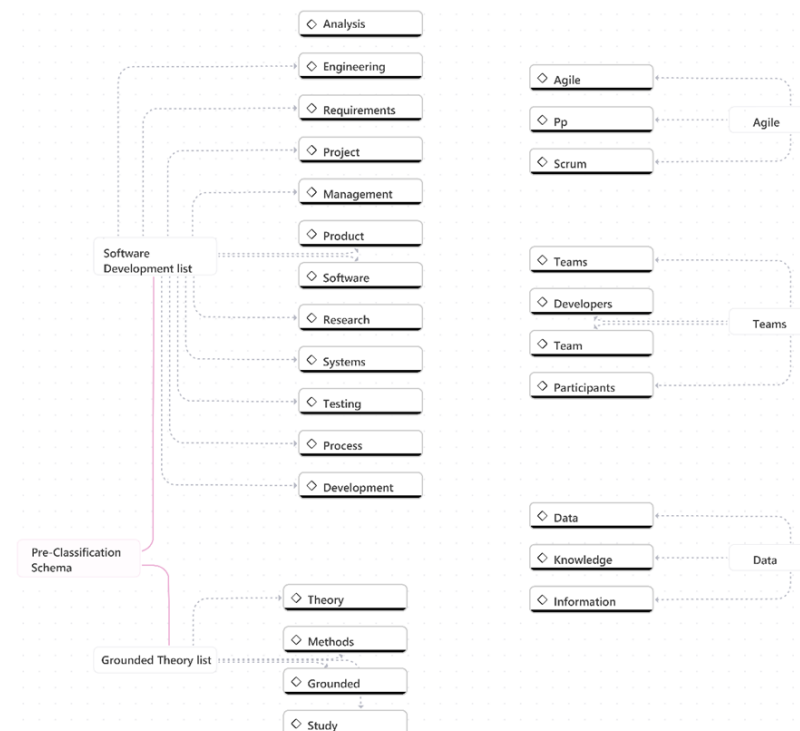


Figure 2. Five categories were identified through selective coding: two from the pre-classification schema and three emergent categories.

Theoretical coding revealed that further in-depth study of the last three categories, {Agile}, {Data}, and {Team}, is illustrated in Figure 1.

The three terms, {Agile}, {Data}, and {Team}, successfully encompassed the concepts and keywords present in the documents that did not fit into the two predefined lists of the pre-classification schema. Each of these terms is grouped into a series of related terms. It is also important to note that these terms were the first to emerge during the analysis and are not necessarily the most suitable. Our focus is on identifying terms that appear with the highest frequency.

3.1.2. Mapping

The three categories of Keywording {Agile}, {Data}, and {Team} are the starting points for the Mapping stage.

Concept {Agile}

Mapping: During the open coding stage, the concept of {Agile} was initially identified based on works explicitly aligned with Agile methodologies; as the analysis progressed to the selective coding stage, it became evident that there was a substantial number of works that did not align with the principles or practices of Agile methodologies. These works exhibited characteristics that diverged from Agile, such as a focus on linear processes, detailed upfront planning, or limited adaptability, which are commonly associated with traditional software development approaches.

To encapsulate these works and provide a structured categorization, a new concept emerged: {Non-Agile}. This concept was developed by systematically examining documents that were explicitly not categorized as Agile during the coding process, ensuring a comprehensive and grounded classification. The {Non-Agile} category is a conceptual counterpart to {Agile}, allowing for a more nuanced analysis of the theoretical interactions between these two opposing paradigms.

Finally, during theoretical coding, the interplay between the categories {Agile} and {Non-Agile} enabled the identification of broader theoretical patterns and relationships. This opposition reflects the spectrum of methodological practices in software development and serves as a foundation for constructing a grounded theoretical framework encompassing the observed approaches. Specifically, the open coding stage introduces the {Agile/Non-Agile} duo, which is further explored in selective coding. This exploration gives rise to a new code, {Plan-driven}, and a refined pairing, {Agile/Plan-driven}. Ultimately, theoretical coding reveals that software development can be approached through two distinct yet interconnected frameworks: {Agile/Non-Agile} and {Agile/Plan-driven}. Notably, the category {Non-Agile} subsumes the {Plan-driven} code. These findings are illustrated in Figure 3. This pairing introduces a novel method of categorizing software development and directly addresses RQ5.

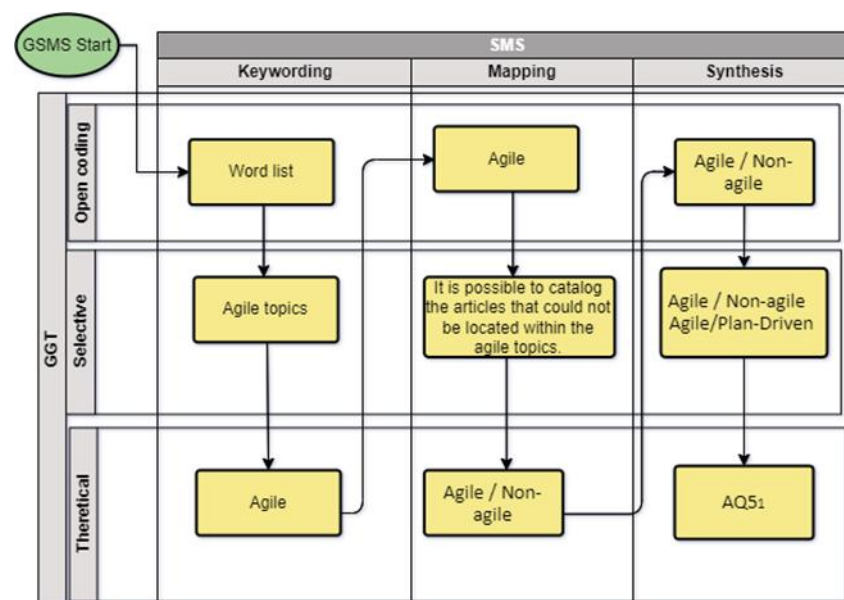


Figure 3. Keywording/Mapping and Synthesis around Agile—Non-Agile topics.

Concept {Data}

In the Mapping stage, the selected terms from open coding are systematically grouped to uncover patterns and relationships. During the selective coding stage, the term {Data} is identified as a central theme and organized accordingly. This organization paves the way for the emergence of a broader concept during theoretical coding: {Data diversity}. This concept highlights the variety and richness of data sources utilized in software engineering research.

The concept of {Data diversity} underscores the importance of incorporating diverse forms of data in software engineering studies. Examples of novel data collection methods include Internet mailing lists [12] and email interviews [13], which expand beyond traditional qualitative approaches commonly used in Grounded Theory (GT). Traditional data sources in GT can be further enriched with artifacts specific to software development

practices, such as programming codes, software methodologies, or tools actively used by practitioners.

This diversity is evident in various examples:

- The debugging process provides insight into error resolution techniques and developer workflows [14].
- Reusable architectures exemplify systematic approaches to software design [15,16].
- The critical role of communication tools, such as those used by participants in Open-Source Software Development (OSSD), is to serve as essential data points for understanding collaborative practices [12].

By integrating traditional and novel data sources, {Data diversity} serves as a robust foundation for analyzing and understanding complex phenomena in software engineering, reinforcing the adaptability of Grounded Theory to this domain.

Concept {Teams}

In the Mapping stage, the selected words from open coding are grouped systematically. Selective coding identifies the term {participants}, also noted in Table 1, and organizes them accordingly. Subsequently, during theoretical coding, a concept emerges referring to the {types of participants}. These participants serve to participate either individually or as part of a {group}.

Open coding entails the identification of {participants}. Selective coding delves into the various interactions among participants and the information about Software Development. Finally, theoretical coding introduces the proposition of {Single participant/Team}, which encompasses multiple types of {participants}, such as programmers, actors, roles, and stakeholders, as well as a variety of {teams}.

3.1.3. Synthesis

The Synthesis resulted in the following key outcomes:

1. **{Agile/Non-Agile}** and **{Agile/Plan-driven}**: These frameworks provide a novel way to categorize software development approaches based on their alignment with Agile principles. The 70 articles were classified into two large groups: 28 papers related to Agile and 42 papers associated with Non-Agile, with **{Plan-driven}** being a subset of Non-Agile.
2. **{Data diversity}**: This concept highlights the importance of incorporating diverse data sources in software engineering research, including traditional and novel data collection methods. The diversity of data is summarized in terms of participants and documents, providing an additional way to categorize software development.
3. **{Single participant/Team}**: This proposition emphasizes the importance of understanding team dynamics and individual contributions in software development. It categorizes participants into individuals and teams, reflecting how they contribute to the development process.
4. These findings provide a comprehensive understanding of the multifaceted aspects of software development, addressing **RQ5** and contributing to the overall goal of categorizing software engineering phenomena using Grounded Theory.
5. By amalgamating the findings, we synthesized different results, such as **{Agile/Non-Agile}**, **{Agile/Plan-driven}**, **{Single participant/Team}**, and **{Data diversity}**, and established connections between these varied outcomes. This Synthesis allows for a comprehensive understanding of the multifaceted aspects of software development.

3.1.4. Verification of Saturation Conditions

In the second iteration, no additional research questions emerge.

$$Q_2 = \left\{ \sum_{i=1}^n RQ_i \right\} = \{RQ_1 + RQ_2 + RQ_3 + RQ_4 + RQ_5 + RQ_6\} \quad (4)$$

where Q_2 is the set of research questions of iteration 2.

$$A_2(k) = \sum_{j=1}^m AQ_j \quad (5)$$

where $A_2(k)$ is the set of Answers to the Questions of iteration 2.

$$A_2(4) = \{\mathbf{AQ4}_1: \{\text{GTe}\}\}; \text{ From iteration 1} \quad (6)$$

where $A_2(4)$ is the set of the Answer to the research question 4 (RQ_4) of iteration 2.

$$A_2(5) = \{\mathbf{AQ5}_1: \{\text{Agile/Non_agile}\}, \mathbf{AQ5}_2: \{\text{data diversity}\}, \mathbf{AQ5}_3: \{\text{Single participant/team}\}\} \quad (7)$$

where $A_2(5)$ is the set of three Answers to the research question 5 (RQ_5) of iteration 2.

AQ5₁: {Agile/Non – agile} is Response 1 to Question 5. Of the 70 articles, 28 were classified as Agile and 42 as Non-Agile.

AQ5₂: {data diversity} is Response 2 to Question 5. It can be summarized in emails, members, documents, etc.

AQ5₃: {Single participant/Team} is Response 3 to Question 5. The participants could be individuals or teams.

Replacing (6) and (7) in (5).

$$A_2(k) = \{\{\mathbf{AQ4}_1: \text{GTe}\} + \{\mathbf{AQ5}_1: \{\text{Agile/Non_agile}\}, \mathbf{AQ5}_2: \{\text{data diversity}\}, \mathbf{AQ5}_3: \{\text{Single participant/team}\}\}\} \quad (8)$$

For this iteration, the six research questions—RQ1, RQ2, RQ3, RQ4, RQ5, and RQ6—remain the same as those defined in iteration 1. Since both the number and set of questions are unchanged in iteration 2, it can be concluded that no new research questions emerged. Therefore, one of the three saturation conditions has been met.

However, new answers to the research questions were identified in iteration 2, which means the second saturation condition has not yet been satisfied [9,10].

3.1.5. Comparison with the Literature

During the Comparison with the literature stage, it is crucial to distinguish between Non-Agile and Agile software within the scientific literature. Although an explicit classification of {Agile/Non-Agile} or {Agile/Plan-driven} was not found in the references, several limited uses of the term **{Non-Agile}** were identified. For instance, traditional software development is considered **{Non-Agile}**, and even within a Non-Agile development environment, Agile requirements engineering techniques are applied [17].

Furthermore, the literature review revealed that information systems educators incorporate the concept of agility into their teaching, identifying approaches that include using Non-Agile methods to teach Agile content [18]. References were also found regarding the application of software development approaches in both Agile and Non-Agile contexts [19], and a positive correlation was established between job satisfaction in teams using both Agile and Non-Agile methodologies [20].

While the literature provides examples of ‘Non-Agile’ development, a more detailed and explicit classification of the terms ‘agile’ and ‘Non-Agile’ remains to be explored, which would allow for a deeper and more meaningful comparison.

Although there are references to both Agile and Non-Agile development, we have not found explicit categorizations of {Agile/Non-Agile} and {Agile/Plan-driven}.

Given that the GT methodology was employed and following the existing documentation, it is reasonable to include data sources and human actors—such as individual participants or development teams—as part of the categorization process at this iteration stage. However, when comparing our emerging categories with those reported in the

literature, no explicit classifications corresponding to dimensions such as **{Single participant/Team}** or **{diversity of data}** were identified. These aspects appear to be underrepresented or implicitly treated, suggesting a potential gap in how human factors and data variability are addressed in existing development software-related taxonomies.

3.2. Third Iteration: Integration of Grounded Theory Elements (GTe) in Software Engineering Classification

This iteration focuses on analyzing the influence of Grounded Theory elements (GTe) in the classification of software engineering. The concept of GTe—a novel classification schema derived from Grounded Theory principles—was introduced in iteration 1 (explained briefly in Sections 2.2, 2.3, and 2.4 and is elaborated in detail in Navas's work [10]. As a direct outcome of iteration 1, research question 6 (RQ6) emerged, emphasizing the necessity of classifying software engineering artifacts and practices according to the GTe framework. Consequently, the selected papers are revisited and re-analyzed from this perspective; in other words, the papers are interviewed again.

Inputs of this iteration include the following.

- The six research questions, which are RQ1 to RQ6.
- The pre-classification schema, which is derived from the previous.
- The original set of 70 selected papers.
- Equation 1, one of the three required to reach saturation, which is fulfilled, as shown below.
- One of the three saturation conditions or equations is reached, confirming partial theoretical saturation between the second and third iterations.

$$Q_2 = Q_3 \quad (9)$$

- Partial responses to RQ5, based on emergent categories such as {Agile/Non-Agile}, {Data diversity}, and {Individual participants/Teams}.
- From iteration 1, a consolidated answer to RQ4, framed through the emerging Grounded Theory elements (GTe).

The analytical procedure follows the same structure established in previous iterations, encompassing Keywording, Mapping, and Synthesis, complemented by the classic stages of Grounded Theory coding: open coding, selective coding, and theoretical coding, as shown in Figure 4.

This phase involved a comprehensive recategorization of software engineering concepts by analyzing the literature derived from the initially selected papers, guided by the Grounded Theory elements (GTe) framework, which was developed in the first iteration.

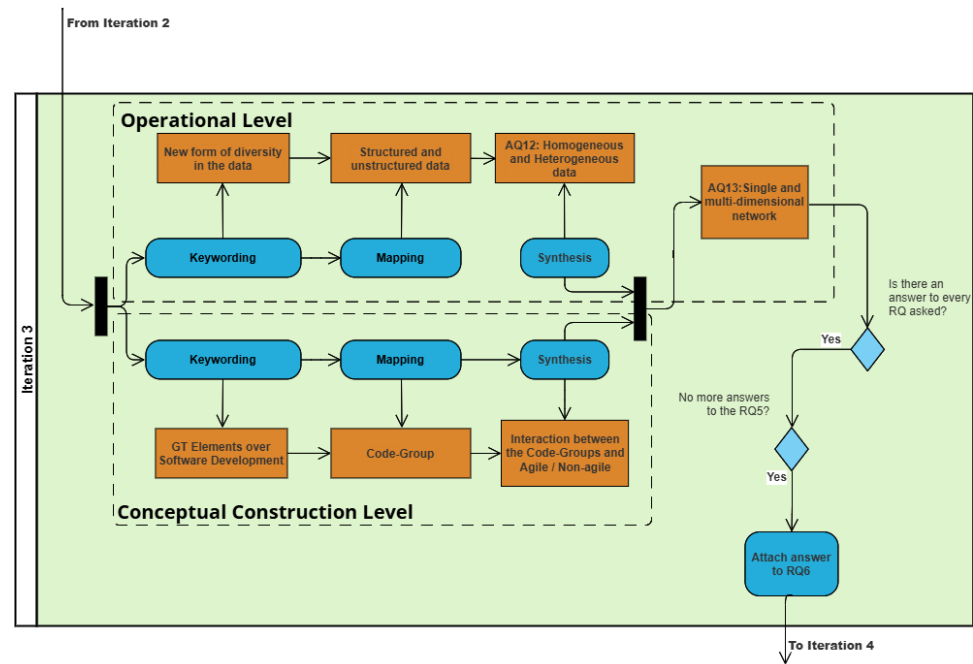


Figure 4. Activity diagram for the third iteration, showing in blue the stages of keywording, mapping, and synthesis; in brown the findings from each stage; and the verification of saturation conditions for this iteration.

3.2.1. Keywording

The Keywording process, undertaken during the open coding phase, utilized the Grounded Theory elements (GTe). This procedure resulted in the recategorization of the GTe into two distinct levels, conceptual and operational, aligning with Morse's [21] conceptual division of Grounded Theory. During the selective coding phase, it structured the analysis into two levels: conceptual and operational [21].

1. **Conceptual Construction (Meta-Level):** This tier focuses on abstract theory-building and encompasses (1) Basic Coding, (2) Master Core Category, (3) A GT of ..., (4) Use GT to ...
2. **Operational Level (Empirical Validation):** This tier ensures methodological rigor through systematic data engagement, including (5) Data, (6) Data Analysis, and (7) Constant Comparative Method.

The theoretical coding reflects Grounded Theory's dual emphasis on generating substantive theory while maintaining empirical fidelity, as advocated by Morse [21].

Conceptual Construction Level

Keywording: The open coding process involved a detailed categorization of Grounded Theory elements (GTe). The first group of GTe comprised four specific elements: Basic Coding, Master Core Category, A GT of ..., and Use GT to It was systematically analyzed each selected article according to these four elements and assigned a unique code to each one. Each code was then explicitly linked to the corresponding sentence, paragraph, or section where its definition appeared in the document.

As discussed by Navas [10], these codes followed a strict formatting structure. For example, codes corresponding to Basic Coding and Master Core Category began with a numeral indicating the number of categories or concepts. For emerging theories, the codes started with the phrases "A GT of ..." or "Use GT to ...", depending on how Grounded Theory was applied in the analyzed article.

It was used the extracted codes and their associated textual segments were used, focused exclusively on themes related to software development, to generate a word cloud

for each of the four GTe codes. These word clouds, which illustrate the frequency and distribution of key terms, are shown in Figure 5 {Basic coding}, Figure 6 {Master Core category}, Figure 7 {A GT of ...}, and Figure 8 {Use GT to ...}, in descending order of word frequency. The following sections present the most frequent words and concepts identified for each code.

In the case of {Basic Coding}, the words in the word cloud were as follows: categories, concepts, category, study, grounded, interview, interviews, theory, knowledge, research, software, core, framework, questions, communication, requirements, companies, domain, Glaser, roles, code, developer, qualitative, researchers, Agile, human, information, pp, role, user, developers, documentation, quality, application, company, models, organization, participation, process, processing, source, systems, tools, and users stands out the most, as depicted in Figure 5.

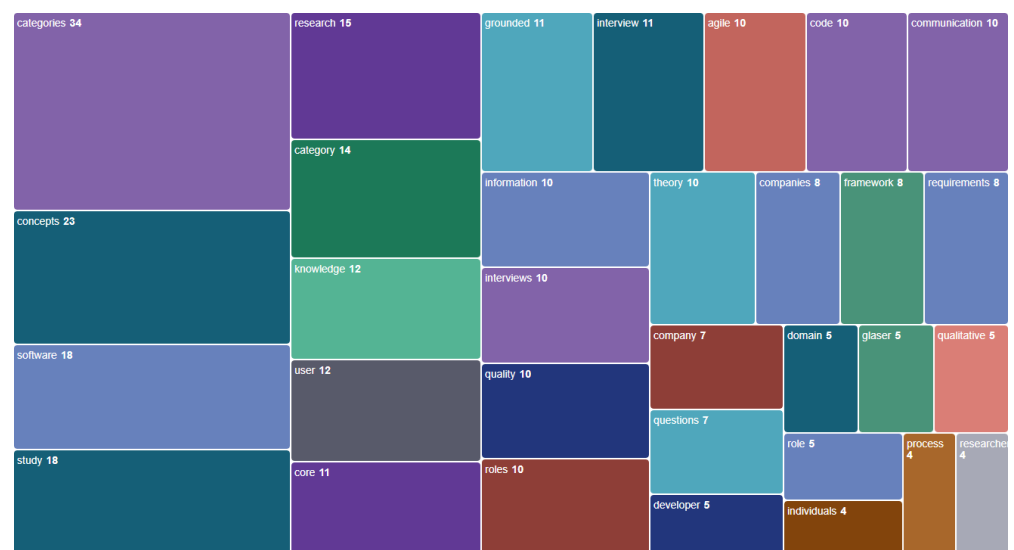


Figure 5. Word cloud regarding Basic Coding. Each rectangle represents a coded word; its size reflects the frequency of occurrence, and the number displayed next to each word indicates its frequency count.

For the {Master Core Category}, the words in the word cloud were as follows: categories, category, concepts, study, interview, interviews, research, grounded, software, companies, Glaser, requirements, core, quality, theory, Agile, code, knowledge, human, information, qualitative, questions, researchers, customer, participants, process, application, domain, role, user, company, organization, and roles. Once again, it is as illustrated in Figure 6.

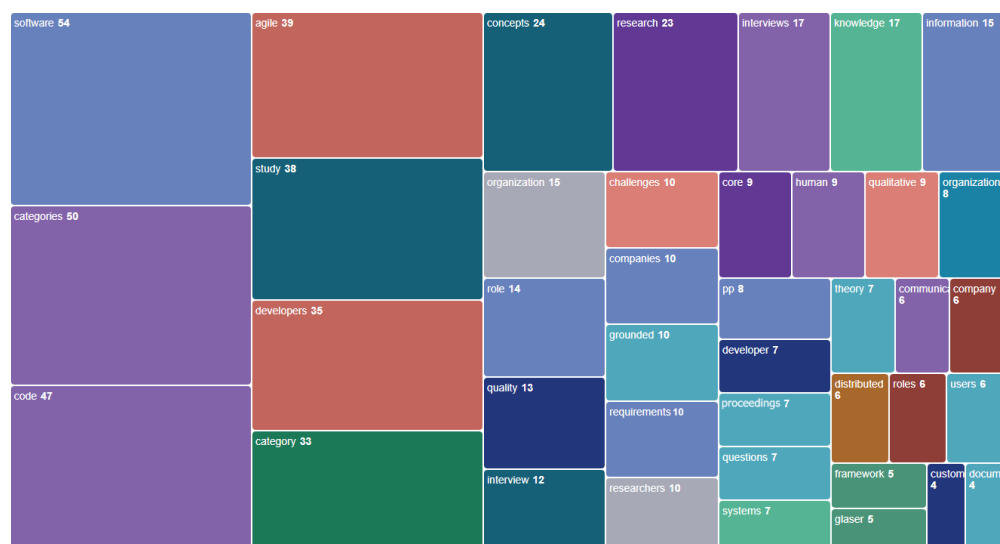


Figure 6. Word cloud regarding the Master Core Category. Each rectangle represents a coded word; its size reflects the frequency of occurrence, and the number displayed next to each word indicates its frequency count.

For {A GT of ...}, the words ordered were: pair programming (Pp), software, research, role, roles, study, developers, Agile, knowledge, requirements, quality, code, organizations, theory, grounded, human, concepts, organization, qualitative, companies, programming, systems, categories, communication, information, company, customer, framework, individuals, interviews, business, challenge, developer, distributed, domain, category, IEEE, user, users, customers, interview, core., one, and can pair programming. Pp appears in 17 documents of the 70 (24%). As can be seen, the one that stands out is Pp, as shown in Figure 7.

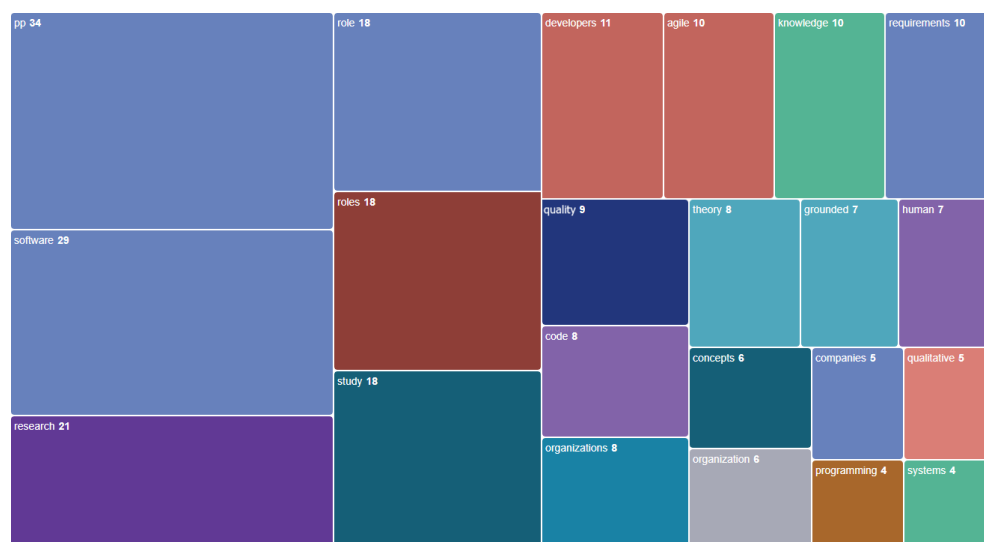


Figure 7. Word cloud regarding A GT of

Finally, for {Use GT to ...}, the words in the corresponding word cloud were study, Agile, software, framework, research, documentation, grounded, source, theory, developers, organizations, requirements, qualitative, challenge, companies, company, core, distributed, interviews, knowledge, people, quality, researchers, role, and users. As can be seen, it is the one that stands out is study, as shown in Figure 8.

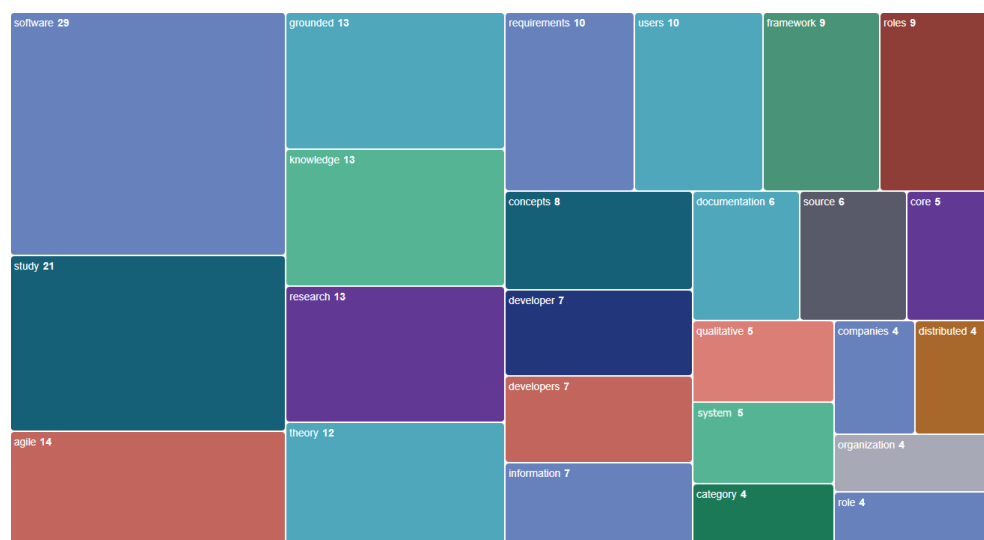


Figure 8. Word cloud regarding Use GT to

Selective coding, in conjunction with the four word lists (Figures 5–8), enables a deeper analysis at this stage and leads to the creation of a new category, the Extended Software Development List (ESDL). Theoretical coding suggests that the ESDL represents an extension of the initial preclassification scheme lists. However, this list is currently taken as is and has not yet been fully classified or characterized to avoid the influence and subjective criteria of the researcher.

Operational Level

Keywording open coding involves a second level that includes by GTe: data analysis and constant comparison of data.

The following shows the words and concepts for each code in the order in which they appear most frequently to least frequently.

Data collection: The words ordered in the word cloud are: {Data}, {interviews}, {questions}, {interview}, {participants}, {process}, {software}, {project}, {agile}, {collection}, {analysis}, {theory}, {development}, {study}, {team}, shown in Figure 9. The data collection is formed from four subcodes in Atlas.TI is related to the three methods of collecting data in GT. It was also grouped the data collection into interviews, questionnaires, and observation. The distribution for each of them is as follows:

Interviews: The words ordered in the word cloud are interviews, questions, data, process, interview, Agile, one, semi-structured, guide, project, analysis, development, conducted, participants, used, and Team.

Questionnaires: The words ordered in the word cloud are ‘Experience,’ ‘interview,’ ‘questionnaire,’ ‘questions,’ ‘analyzed,’ ‘participants,’ ‘results,’ and ‘sample.’

Observation: The words ordered in the word cloud for this topic are Documents, form, requirements, studied, flow, textual, algorithms, data, free-flow, level, Mapping, papers, process, prototyping, and size.

The {data source} could be participants, documents, and artifacts. The participants can be individuals or groups, commonly known as teams.



Figure 9. Word cloud regarding data collection.

Data analysis The words ordered from the greater to lower in the word cloud for this topic are {Data}, {Coding}, {categories}, {analysis}, {theory}, {open}, {codes}, {grounded}, {concepts}, {process}, {category}, {interview}, and {research}, as shown in Figure 10.

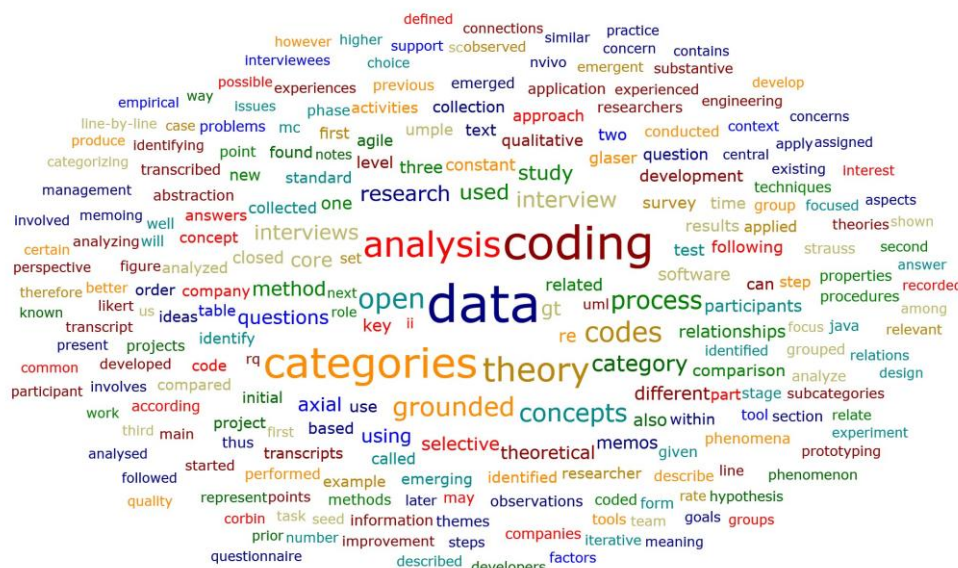


Figure 10. Word cloud regarding data analysis.

These word clouds are associated with GTe, along with the GT list from the pre-classification schema, resulting in the {Extended Grounded Theory List (EGTL)}. These are additional concepts that the open coding stage contributes.

The first category aligns closely with typical practices in Grounded Theory research and includes studies that rely on standard qualitative data collection methods. Specifically, this group comprises papers that mention the use of interviews (51 papers), questionnaires (39 papers), and observations (13 papers).

The second classification involves examining concepts and terms related to data collection methods, including interviews, questionnaires, and observations. These methods

constituted the primary empirical sources, including semi-structured interviews, mailing lists, and others. These data sources may exist in either structured or unstructured formats, giving rise to the second classification —the conceptual distinction between {structured data} and {unstructured data} —and revealing an initial observation: these expressions are not commonly found in the lexicon of software engineering. For example, the term {structured data} appears only once in the SWEBOK, specifically in the context of data structures. In contrast, the word “structure” is widely used across various topics, including structured design, structured analysis, structured programming, structured software, structured libraries, SQL (Structured Query Language), and structured tools within computer-aided software engineering (CASE). This pattern suggests that, although the notion of “structure” is deeply embedded in Software Engineering discourse, the explicit distinction between structured and unstructured data has not traditionally been part of its conceptual vocabulary, suggesting a potential gap between software engineering terminology and data-centric research perspectives.

Selective coding allows the establishment of two categorizations: previous {interviews, questionnaires, and observations} and {structured/unstructured data}. In theoretical coding, we discovered a novel form of diversity within the data.

3.2.2. Mapping

Conceptual Construction Level

At the Conceptual Construction Level, particularly during the open coding stage of the Mapping phase, the ESDL was linked to three key memos: *Basic Coding*, *Master Core Category*, and *Emerging Theory*. These memos were mapped together to examine their relationships within the context of Software Development.

During selective coding, it was identified, and it grouped related concepts, forming a code-group that reflects the structural relationships among elements of Software Development. By systematically Mapping these elements, we established how they can be classified, which led to theoretical coding and the creation of a new classification termed the {Software Development Code-Group (SDCG)}. The SDCG emerges directly from the ESDL categorization and characterization process.

The Software Development Code-Group (SDCG) includes the following categories:

- Architecture;
- Agile transformation process (ATP);
- Company;
- Innovation;
- Knowledge sharing;
- Model-metamodels-UML;
- Pair programming (PP);
- Requirements;
- Software Development process;
- Team;
- Tool;
- Testing;
- Open-source Software Development.

The diversity of the SDCG components initially generated substantial debate among research participants, as the items appeared to lack an explicit or intrinsic relationship. However, Grounded Theory (GT)—and particularly the Glaserian Grounded Theory (GGT) embedded in the GSMS framework—requires that results emerge solely from data analysis rather than from researcher assumptions. Consequently, we accepted and consolidated this list as it emerged inductively from the GSMS process.

The Synthesis Stage is discussed in detail in Section 3.2.3, where the SDCG is analyzed in conjunction with the results obtained from the Operational Level.

Operational Level

The Operational level: Mapping open coding, involves Mapping the various data sources, including the diversity of data generated by the diverse range of participants, as well as the diversity of data types established through structured, semi-structured, and unstructured data. Selective coding: It is used to establish various categories for this diverse data type. Theoretical coding shows this diversity in three types.

The Operational Level: Mapping open coding focuses on systematically Mapping the different data sources involved in the study. This process integrates the variety of data generated by participants, encompassing the diversity of data types, including structured, semi-structured, and unstructured data. During selective coding, it was established that categories represent this heterogeneity, while theoretical coding classifies the diversity into three main types:

1. The participant and the information they generate;
2. Structured and unstructured data;
3. Data from the interviews, questionnaires, and observations.

Just as a physician cannot be separated from their stethoscope, software engineering remains inseparable from its artifacts.

In Synthesis open coding, it was moved from the concepts extracted from these three data forms toward selective coding, which integrates and refines them into higher-order categories. Selective coding recognizes that data in Grounded Theory (GT) primarily consists of unstructured text [22]. Open-ended interviews often generate unstructured and ambiguous narratives [14]. And their transcripts naturally follow an unstructured format [23]. Consequently, these transcripts constitute a substantial body of unstructured data that gains meaning only when analyzed through GT procedures [24].

Although modern software development typically associates unstructured data with machine learning and big data analytics [25], GT has long provided a systematic framework for interpreting unstructured and heterogeneous information within qualitative inquiry.

Theoretical Coding

The preceding stages reveal the need to establish a new classification of Grounded Theory (GT) data within the field of software engineering. The data was obtained through interviews, questionnaires, and observations, which demonstrate a dynamic interaction between data collection processes and the inherent heterogeneity of the data. Based on these findings, a new code {heterogeneous data} was introduced to broaden the analytical scope of GT. This new category extends the theoretical framework by explicitly acknowledging the diversity of data sources and formats involved in software engineering research. The analysis presented in [22] supports this refinement.

3.2.3. Synthesis

The most significant findings emerge from the two previously developed levels: the Conceptual Construction Level and the Operational Level.

At the Conceptual Construction Level, Synthesis is carried out for each element identified within the {Software Development Code Group (SDCG)}, considering its relationship with the results obtained across all iterations.

At the Operational Level, Synthesis focuses on the integration and interpretation of all empirical data. Finally, the results obtained from the integration of both levels are analyzed. This integration consolidates and reorganizes prior findings related to the SDCG,

exploring their relationships with agility, heterogeneous data, the fundamental elements of Grounded Theory (GT), and the notion of {bridges}, shown in Figure 11.

The ultimate goal is to provide a cohesive and synthetic view of how each construct emerged and how the various findings interrelate, serving as a key analytical unit for categorizing software engineering phenomena through the lens of Grounded Theory.

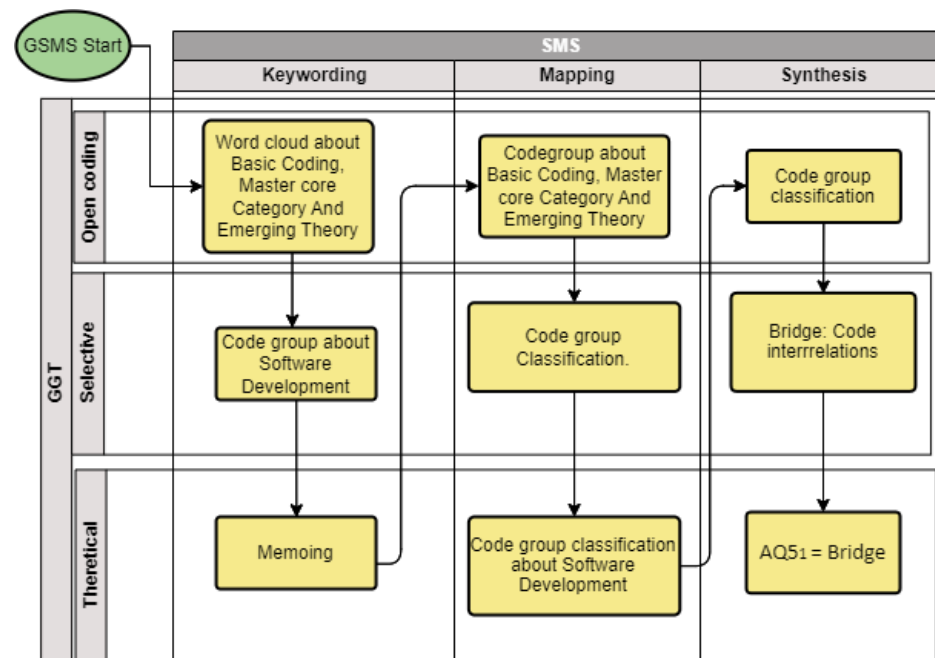


Figure 11. Keywording, Mapping, and Synthesis: bridging to the bridge.

Relation to Agile and Non-Agile Topics

This subsection deepens the previous analysis of the relationship between the Software Development Code Group (SDCG) and both Agile and Plan-driven approaches by offering a more comprehensive interpretation of the data.

It initiated the open coding process to map the SDCG to topics related to Agility, which emerged as one of the responses to the AQ5 equation presented in Section 3.1.4. During selective coding, 28 articles were identified addressing Agile practices. It organized these studies into the following code groups:

- {Agile transformation process (ATP)}.
- {Agile architecture}.
- {Teams: Agile teams, owner teams, distributed teams, etc.}.
- {Pair programming}.
- {Knowledge sharing}.

In contrast, it categorized 42 articles focused on Plan-driven approaches into the following code groups:

- {Architecture}.
- {Company}.
- {Model–metamodels–UML}.
- {Tools}.

The Model–Metamodels–UML group encompasses topics ranging from fundamental modeling practices, such as UML, to more abstract representations, such as metamodels. Several of these studies also examine the human dimension and contextual environment. The tools group gathers works proposing methodological enhancements and innovative tool designs for software development.

Certain code groups, such as *architecture*, *Teams*, and *tools*, appear in both methodological paradigms, indicating their cross-cutting nature. Others, such as ATP and *Knowledge Sharing*, emerge exclusively within the Agile context, underscoring their close association with Agile frameworks.

During theoretical coding, the interactions were analyzed between these code groups and the overarching themes of Agility and Non-agility. Among the total corpus, 28 papers (40%) explicitly address agility-related topics.

The data reveal that specific categories, particularly *Teams* and *tools*, are transversal, appearing in both Agile and Non-Agile contexts. This finding suggests that collaboration and tool usage remain fundamental dimensions of software development, independent of the chosen methodological paradigm. For instance, the *Teams* category appears in both types of studies (14 Agile and 8 Non-Agile), confirming that teamwork constitutes a universal component of development practice.

Conversely, categories such as ATP and Knowledge Sharing appear exclusively in Agile-related studies, emphasizing that specific practices and processes are intrinsic to Agile methodologies.

As summarized in Table 2, these patterns demonstrate that the SDCG functions as a conceptual interface between contrasting development paradigms. This bridging role positions the SDCG as a foundation for integrating divergent methodological perspectives—a concept explored further in subsequent sections

Table 2. Agility and Non-agility vs. code groups.

	Agility	Non Agility
Architecture	1	3
ATP	7	
Company	1	7
Innovation		1
Knowledge sharing	6	
Model–metamodels–UML	2	15
Pair programming	2	1
Requirements	1	7
Software Development process	9	7
Teams	14	8
Tools	4	4
Testing	2	3
Open-source Software process	0	3

Relation to Heterogeneous Data

Heterogeneous data establishes a new classification when applying the GT in software development, distinguishing between data with a schema and data without one. We call them structured, semi-structured, and unstructured data.

Several SDCG elements—pair programming, tools, testing, and Team—are discussed based on data collected through diverse methods, including interviews, observations, documentation, and software artifacts. It supports the theoretical development of heterogeneous data as a category, showing how different data types enrich the understanding of sociotechnical systems.

It follows that the SDCG are not just thematic groupings but represent convergence points between various forms of empirical evidence.

Before continuing, it is going to be defined as structured and unstructured data. Structured data has a schema that defines it, and unstructured data does not [26] found in Lidong Wang's BigData studies [25].

Interviews in GT are usually classified as semi-structured because the interview has a schema {structured data} that corresponds to the planning of the questions, and what is unstructured is found in the answers obtained. Figure 12 shows an example of the Synthesis of the {unstructured data}.

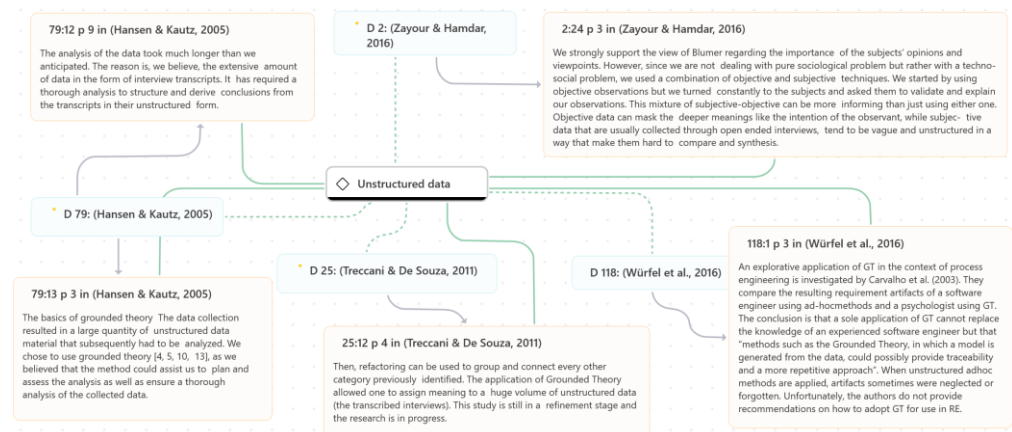


Figure 12. Synthesis of unstructured data derived from the analysis of [14,23,24,27].

During open coding, the relationships were analyzed among previously identified concepts, categories, and propositions to uncover a framework of interconnections. These relationships emerged between Agility and Non-agility, between different types of collected data, and between theoretical models and their industrial applications. It was identified that relationships among research participants, such as teams, roles, and actors, as well as across various fields of software development, exist between Grounded Theory (GT) and other research methodologies.

In the selective coding stage, it was determined that these interconnections are not incidental; instead, they represent a fundamental characteristic of GT when applied to software development.

During theoretical coding, this defining characteristic of GT revealed its capacity to build authentic *bridges* among diverse fields, participants, institutions, and methodological approaches within software development. Each analyzed article provides evidence of these interrelationships and conceptual bridges, confirming the integrative nature of GT in this research context.

Relation to Grounded Theory Elements (GTe)

Multiple SDCG elements align directly with the Grounded Theory elements (GTe) as defined in iterations 1 and 3. For example:

- Requirements, Software Development process, and model–metamodels (UMLs) appear in both {Basic Coding} and {Master Core Category}.
- Pair programming (PP) stands out as a dominant topic in the {A GT of ...} theories, with 17 mentions.

It confirms that the SDCG are not merely context-driven codes, but reflect theoretical patterns identified at different GT coding levels. This relationship strengthens the internal validity of SDCG as grounded analytical categories.

Shows a direct relationship between the Basic Coding and Master Core category because there is a coincidence in the number of topics in both GT elements.

- Engineering models and metamodels have 10.

- Software testing has 7.
- Software requirements have 6.
- The Engineering process has 5.

It was also observed that the number of theories is the most significant, with models, metamodels, methods, and UML being the most notable.

Bridges

A cross-analysis of the SDCG reveals that several components act as bridges connecting traditionally segregated domains within software development. For example, Model–Metamodels, UML serves as a bridge between theoretical foundations and UML-based tools; Team connects individual roles with group dynamics; and tool links Grounded Theory (GT) data sources to developer-oriented technologies.

A bridge is conceptualized as an emergent relational construct that links categories or concepts that have previously been analytically distinguished within the theorizing process. This relational construct operates as an articulating mechanism that enables the integration, transition, and co-interpretation across distinct conceptual domains, while preserving the properties and analytical identity of the entities it connects.

This bridging behavior highlights the multidimensional nature of the SDCG and underscores its potential to elucidate complex interrelationships across software engineering processes. It was identified multiple instances of such bridges spanning diverse subfields of software development were identified. These include studies that bridge heterogeneous data and human actors, those addressing sociotechnical aspects, works linking theoretical models with their industrial applications, and others connecting software engineering research with corporate practice. This constitutes the answer $A_3(3)$ to RQ3, following some examples:

- A bridge between Agile methodologies and cybersecurity, proposing the integration of a novel cybersecurity model within Agile environments [28].
- A bridge connecting open-source repositories on GitHub with the identification of professionals responsible for specific code contributions, analyzing how developers engage with the technical debt arising from their work [29].

Overall, these bridges connect:

- Methodological paradigms (agile vs. Non-Agile).
- Levels of abstraction (theory vs. industrial application).
- Human actors (developers, designers, users).
- Data types (heterogeneous and multi-source data).
- Methodological approaches (qualitative methodologies).
- Software development domains.
- Multidimensional topics.

Their implications for understanding the integrative dynamics of software engineering research will be explored in depth in Section 4.

3.2.4. Verification of Saturation Conditions

The application of Grounded Theory (GT) to software development processes and tasks was explicitly evaluated using the Grounded Theory elements (GT_e) framework, enabling the assessment of its methodological correctness beyond a merely instrumental use. The analysis indicates that most of the reviewed studies adhere to the core principles of GT, including the use of empirically grounded data relevant to the software engineering domain, the systematic application of open, selective, and theoretical coding, and the consistent use of the constant comparison method. Moreover, the recurrent presence of elements such as {Basic Coding}, {Main Category}, and explicitly articulated theories expressed as {A GT of ...} or {Use GT to} demonstrates that GT is applied not simply as a

qualitative analysis technique, but as a comprehensive theory-generation process aimed at explaining socio-technical phenomena in software development.

The results of the current iteration further confirm that GT is correctly applied when it is coherently integrated with software engineering-specific processes and tasks. In particular, the findings reveal a consistent adaptation of GT to contexts such as Agile teams, plan-based processes, and the use of diverse data sources, which were previously identified in iteration 2 in response to RQ5. At the operational level, the functioning of the GT elements—{Data}, {Data Analysis}, and {Constant Comparison}—leads to the emergence of higher-level analytical outcomes, including the Software Development Code Group (SDCG), heterogeneous data management practices, and the identification of bridges between actors, practices, and tools. These outcomes constitute the response to RQ6 in iteration 3. Consequently, the empirical evidence analyzed supports the conclusion that GT, when applied in accordance with its constituent elements and within the GSMS framework, is methodologically sound for the study of software development processes and tasks, thereby providing an affirmative answer to RQ2.

In the third iteration, no additional research questions emerge.

$$Q_3 = \left\{ \sum_{i=1}^n RQ_i \right\} = \{ RQ_1 + RQ_2 + RQ_3 + RQ_4 + RQ_5 + RQ_6 \} \quad (10)$$

where Q_3 is the set of research questions of iteration 3.

$$A_3(k) = \sum_{j=1}^m AQ_j \quad (11)$$

where $A_3(k)$ is the set of Answers to the Questions of iteration 3.

$$A_3(4) \equiv A_2(4) \equiv A_1(4) = \{ \mathbf{AQ4}_1: \{\text{GTe}\} \} \quad (12)$$

where $A_3(4)$ is the set of the Answer to the research question 4 (RQ_4) of iteration 3.

$$A_3(5) \equiv A_2(5) = \{ \mathbf{AQ5}_1: \{\text{Agile/Non_agile}\}, \mathbf{AQ5}_2: \{\text{data diversity}\}, \mathbf{AQ5}_3: \{\text{Single participant/team}\} \} \quad (13)$$

where $A_3(5)$ is the set of the Answer to the research question 5 (RQ_5) of iteration 3.

$$A_3(6) = \{ \{ \mathbf{AQ6}_1: \{(\text{SDCG})\} \}, \{ \mathbf{AQ6}_2: \{\text{Homogeneous and Heterogeneous data}\} \}, \{ \mathbf{AQ6}_3: \{\text{Bridge}\} \} \} \quad (14)$$

where $A_3(6)$ is the set of the Answer to the research question 6 (RQ_6) of iteration 3.

$$A_3(1) = \{ A_3(5) + A_3(6) \} \quad (15)$$

where $A_3(1)$ is the set of the Answer to the research question 1 (RQ_1) of iteration 3.

$$A_3(2) = \{ \mathbf{AQ2}_1: \{ \mathbf{GT} \rightarrow A_3(5) \} + \mathbf{AQ2}_2: \{ \mathbf{GT} \rightarrow A_3(6) \} \} \quad (16)$$

where $A_3(2)$ is the set of the Answer to the research question 2 (RQ_2) of iteration 3.

$$A_3(3) = \{ \mathbf{AQ3}_1: \{\text{Bridge between a theoretical model and its industrial application}\} \} \quad (17)$$

where $A_3(3)$ is the set of the Answer to the research question 3 (RQ_3) of iteration 3

$$A_3(k) = \{ A_3(1) + A_3(2) + A_3(3) + A_3(4) + A_3(5) + A_3(6) \} \quad (18)$$

where $A_3(k)$ is the set of Answer Questions of iteration 3.

3.2.5. Comparison with the Literature

It is interesting to identify the literature reviews that incorporate heterogeneous data sources. In the study [30] on non-functional requirements, the literature was followed by interviews and surveys with specialists. So, data heterogeneity arises between the data obtained from the analyzed scientific articles and that collected through surveys and interviews [30].

3.3. Fourth Iteration

The fourth iteration begins with two of the three necessary conditions for saturation already fulfilled.

The first condition is that no new research questions have emerged, and the second condition is that all research questions have at least one answer.

At this stage, it is therefore necessary to verify whether additional answers can still emerge for any of the existing research questions.

The inputs for this iteration are

- The six research questions identified: RQ1, RQ2, RQ3, RQ4, RQ5, and RQ6.
- The pre-classification schema.
- The 70 originally selected papers.
- The set of answers associated with RQ1–RQ6 obtained in previous iterations.

In terms of Keywording, no additional word resources are available in this iteration. No new word lists, word clouds, or other elements exist from which further Keywording and Mapping could be initiated.

Related to Synthesis, the third saturation condition is fulfilled in this iteration, as no new answers can be identified for any research question. This situation is formally expressed as

$$A_4(i) = A_3(i) \quad \text{for } i = 1, 2, 3, 4, 5, 6 \quad (19)$$

This equality indicates that the set of answers in the fourth iteration is identical to that of the third iteration, confirming that no additional answers can be obtained. As a result, the third and final saturation equation of the GSMS process is satisfied.

Verification of Saturation Conditions

Table 3 summarizes the progression of the GSMS iterations and the fulfillment of the three saturation conditions across iterations.

Table 3. Summary of iterations and fulfillment of saturation conditions. For each research question (RQ) and its associated answers (AQ), it indicates the specific iteration in which each saturation condition was satisfied, using check marks (✓).

Saturation Condition	RQ and Emerging Answers	Iteration			
		First	Second	Third	Fourth
First condition (No new RQs have emerged.)	RQ1	✓			
	RQ2	✓			
	RQ3	✓			
	RQ4	✓			
	RQ5	✓			
	RQ6	✓			
	Achieved		✓		
Second condition (All RQs have at least one answer.)	AQ1			✓	
	AQ2			✓	
	AQ3			✓	
	AQ4	✓			
	AQ5		✓		
	AQ6			✓	
	Achieved			✓	
Third condition (There are no additional answers to any RQ.)	AQ1				✓
	AQ2				✓
	AQ3				✓

AQ4	✓	
AQ5		✓
AQ6		✓
Achieved		✓

The first saturation condition (no emergence of new research questions) is satisfied early, during the first iteration, for all RQs.

The second saturation condition (each research question having at least one answer) is satisfied progressively across iterations and is fully achieved by the third iteration.

The third saturation condition (no emergence of new answers for any research question) is only satisfied in the fourth iteration, indicating that theoretical saturation has been fully reached at this point.

The final row of each condition block (“Achieved”) indicates the iteration in which the corresponding saturation condition is completely fulfilled.

4. Discussion

This study establishes conceptual relationships between the SDCG and the methodological paradigms of software engineering—namely, the dichotomies {Agile/Non-Agile} and {Agile/Plan-driven}. The analysis highlights the inherent heterogeneity of the data employed in Grounded Theory (GT) research, a term often associated with Big Data but relevant here given the human and socio-technical nature of GT data. Since GT treats all elements arising from human-artifact interactions as data, this diversity poses both a methodological challenge and a source of theoretical richness.

Integration under the unifying concept of the Bridge provides substantial theoretical insight. In total, 25 papers were identified that fit one or more of the bridge types described earlier, with 11 exhibiting multidimensional bridging characteristics, spanning several analytical dimensions simultaneously. Altogether, 31 articles (44.3%) demonstrate the presence of bridging phenomena that connect diverse methodological, conceptual, and practical domains. These bridges link Agile and plan-driven processes, heterogeneous data-collection strategies, different actors in software development ecosystems, theoretical models, industrial practice, and the interplay between Grounded Theory and complementary research methodologies.

The data analysis further reveals that seven of the reviewed papers explicitly address high levels of data diversity and heterogeneity through various methods of data collection, including semi-structured interviews [31], participant observation [31], and online questionnaires [32,33]. Several studies adopted a multistage approach to data collection, beginning with pre-questionnaires to obtain participant background information, followed by the review of archival materials—formal or informal—and concluding with semi-structured interviews or focus groups [34,35]. For example, Lubin [34] employed a two-stage process combining think-aloud sessions with semi-structured interviews to understand programmers’ cognitive processes. Similarly, Ardo et al. [28] used semi-structured interviews with 23 Agile practitioners and validated their theoretical model through focus groups with cybersecurity experts. Other studies examined secondary sources such as repositories, including Common Weakness Enumeration (CWE) and Common Vulnerabilities and Exposures (CVEs) [36]. Collectively, these examples demonstrate that GT-based research in software engineering not only embraces data heterogeneity but also depends on the timing and contextual integration of data collection activities to ensure theoretical robustness.

Let us delve deeper into the types of bridges based on the seven categories, shown in Table 4.

Table 4. Bridge type.

Bridge Type	Description	Example
Methodological Bridge	Connects Agile and Non-Agile paradigms	Hybrid development environments
Human–Technical Bridge	Links actors and artifacts	Developers interacting with tools
Human Factor Bridge	Connection and integration among diverse actors and/or teams	Collaboration between developers and designers
Research Methodology Bridge	Connects Grounded Theory with other research methodologies and paradigms	GT combined with Action Research or Method Engineering (e.g., Grounded Method Engineering)
Data Bridge	Integrates structured and unstructured data	Interviews + repositories
Theory–Practice Bridge	Connects theoretical GT models with industrial applications	DevOps GT studies
Multidimensional Bridge	Integrates multiple analytical axes	Agile Transition Process

A key finding concerns the variety of bridges across methodological paradigms. Several papers demonstrate interactions between Agile and plan-driven approaches [37–41]. Some organizations apply both simultaneously [37,39], while others selectively integrate Agile principles into plan-driven environments or vice versa [40]. In certain cases, hybrid development strategies leverage the strengths of both paradigms to optimize productivity and adaptability [38,41]. These findings indicate that GT is particularly effective for capturing hybrid methodological realities and the coexistence of contrasting development logics within software projects. Additionally, bridges have been observed between User eXperience (UX) and Agile approaches, especially within the context of Agile UX Integration Practices [32]. Where UX design cycles are synchronized with Agile sprints to enhance collaboration and usability outcomes.

Another pattern involves bridges between theoretical models and their industrial applications. Studies have demonstrated that GT can serve as a framework to connect conceptual research with professional practice in software engineering [42–45]. For example, GT has been used to design and validate surveys with practitioners in requirements engineering [43], to predict and explain the adoption of software processes in small enterprises [45], and to improve security practices in organizations preparing to implement DevOps [44]. Similarly, research linking architectures and smart city mobile applications (SCMA) demonstrates GT's capacity to ground theoretical constructs in practical development contexts [33].

Human factors constitute a further dimension of bridging. GT facilitates the exploration of collaboration and integration among diverse actors, such as developers, designers, users, and multidisciplinary teams [37,46,47]. These studies reveal bridges between developers and designers [46], between programmers and artists [47], and across teams that include testers, architects, and project managers [37]. Moreover, the human-centered focus of GT enables an understanding of creativity, innovation, and communication dynamics in both traditional and open-source development environments [12]. The interaction among three actors in Open Source Software Development (OSSD) projects involves the leader, team members, and users [12].

Data heterogeneity emerges as a methodological bridge that links diverse sources, formats, and contexts. Several studies extend traditional GT data collection by incorporating programming code, software artifacts, and communication logs as empirical evidence [12,14–16,22,48–52]. Such heterogeneous data enable the reconstruction of complex development phenomena that transcend textual narratives. Stol, Ralph, and Fitzgerald [22] emphasize that managing large volumes of heterogeneous data is a significant challenge but also an opportunity for theoretical integration. Examples include GTs of human model-merging activities [48], pair-programming interactions [49], confidential business documentation on

Agile practices [50], and communication exchanges between students [51]. These cases confirm that software engineering data are inherently multimodal, requiring analytical frameworks that can integrate technical, behavioral, and organizational evidence.

Methodological bridging between GT and other research paradigms also appears recurrently in the analyzed literature. Studies combining GT with action research, lateral thinking, controlled experimentation, and method engineering demonstrate that GT's flexibility supports both qualitative exploration and systematic validation [53–56]. The emergence of Grounded Method Engineering (GME) [56] exemplifies this Synthesis, as it builds methods and frameworks grounded in empirical data while retaining theoretical rigor. This integrative potential reinforces GT's role as a methodological mediator across empirical and theoretical dimensions.

In the domain of software engineering, GT has been applied to bridge specific subareas, such as requirements engineering and systems architecture [57], as well as to propose iterative processes, such as Grounded Requirements Engineering (GRE) and *GT2UC* transformations [27]. These applications demonstrate GT's capability to facilitate transitions from conceptual insights to standardized modeling practices. Collectively, these bridges reveal that GT can manage diverse and even opposing, thematic relationships, leading to the Synthesis of new theoretical categories that address the research questions posed at the outset of this study.

Particularly noteworthy is the identification of multidimensional bridges, where multiple axes of analysis interact simultaneously. The Agile Transition Process (ATP) exemplifies this multidimensionality, encompassing transitions in software development practices, team structures, management approaches, reflective activities, and organizational culture [58–63]. Each dimension evolves independently, reflecting the uniqueness of every Agile transformation. Beyond the Agile context, similar multidimensional models emerge in studies on technical debt [64], software quality [37], and knowledge sharing in Agile teams. The Internet-based software development has given rise to a new class of processes distinct from traditional software engineering. Three driving dimensions — competitive environment, lack of experience, and time-to-market pressure — have made quality negotiable and triggered the emergence of “extreme” practices [65]. The adoption of software process standards (ISO/IEC 29110) in micro-enterprises is represented by bridges that include dimensions such as stimuli (awareness and acceptance), organizational conditions, and the application of best practices [66]. Other works identify multidimensional frameworks spanning business vision, project approach, value propositions, stakeholder groups, and contextual factors [67]. These studies collectively indicate that software engineering processes are inherently multidimensional, shaped by interactions between technical, organizational, and human components [68–75].

Finally, multidimensionality also characterizes sociotechnical processes in both Agile and Non-Agile environments. In Agile contexts, GT conceptualizes development teams and the Agile Transition Process (ATP) as complex adaptive sociotechnical systems, where social and technical dimensions coevolve dynamically [14,58–60,76,77]. Conversely, Non-Agile contexts also exhibit sociotechnical bridging, as evidenced in studies on change request management in small companies [78], debugging practices in enterprise IDEs [14], and trust contagion in distributed teams [79]. Additional research connects use case modeling with theory development for demand-response systems [80] and explores coordination mechanisms in software security patch management [70]. Collectively, these examples confirm that both Agile and Non-Agile processes exhibit inherent multidimensionality, with social, technical, and organizational elements converging.

The notion of saturation discussed throughout the GSMS iterations should be interpreted strictly within the methodological boundaries of Grounded Theory. In this study, saturation refers to the stabilization of categories and answers within the analyzed corpus,

as operationalized through the GSMS formal conditions. It does not imply completeness of knowledge in software engineering practice, nor does it suggest that engineering artifacts themselves reach epistemological closure.

5. Threats to Validity

This study faces several threats to validity when viewed through the lens of classic (Glaserian) GGT and within the context of software engineering research artifacts and processes.

5.1. Construct Validity

The analysis is grounded in 70 peer-reviewed primary studies identified via the GSMS protocol. While methodical, this constitutes a protocol-bounded sample rather than theoretically sampled material in the strict Glaserian sense. Consequently, the data may not cover sufficient variation across domains (e.g., safety-critical vs. web systems), development paradigms (e.g., Agile, DevOps, model-driven), or organizational settings (e.g., startups vs. regulated enterprises) to fully establish conceptual fit. This delimitation threatens the fit and relevance of emergent categories—particularly the bridge construct and its typologies—by potentially underrepresenting negative cases or boundary conditions. Although we pursued constant comparison within the available corpus, theoretical sampling across additional, diverse settings is needed to improve conceptual density and delimiting.

Mitigation and residual risk. We documented the GSMS selection decisions and used memos to flag conceptual gaps; however, without directed theoretical sampling beyond the initial corpus, some variation remains unobserved, constraining the fit and relevance of the theory to wider SE contexts.

5.2. Internal Validity

The initial keyword filtering employed frequency-based thresholds from descriptive statistics. From a Glaserian standpoint, this introduces a risk of forcing (i.e., prestructuring the data space before categories are allowed to emerge). While we subsequently applied open coding, constant comparison, and iterative memoing to counteract this risk, the early filter may have suppressed weak signals associated with nascent or low-frequency conceptual indicators—especially for early concept identification related to bridges. This constitutes a threat to emergence and to identifying latent patterns that do not manifest strongly in term frequencies.

Mitigation and residual risk. We revisited the filtering decisions during selective coding and theoretical coding (e.g., considering coding families such as Cause–Effect, Strategy, Process, Context–Condition–Consequence). Nonetheless, the possibility of missed early indicators persists, particularly for subtle bridge mechanisms linking SE artifacts (e.g., requirements–architecture, architecture–test, dev–ops pipelines).

5.3. External Validity

The bridge construct and its typologies are analytically derived from the literature and conceptually validated within the reviewed studies. From a Glaserian perspective, the theory's claims hinge on theoretical saturation and modifiability. Saturation is not yet demonstrated across industrial field settings; thus, the current formulation should be treated as provisionally delimited to the literature-reported contexts. In Glaser's criteria, a theory should fit the data, work (explain the main concern and how it is resolved), be relevant (to those concerned), and be modifiable (open to refinement as new data vary the pattern). Our current status supports fit/work within the corpus but requires further variation to substantiate relevance and modifiability across diverse software lifecycles and socio-technical environments.

Mitigation and residual risk. We articulate the conditions, strategies, and consequences under which bridges were observed in the literature and propose operational proxies (e.g., trace links, dependency graphs, workflow handoffs) for future evaluation. Until industrial theoretical sampling and field validation are conducted, transferability to real-world projects remains a threat.

5.4. Interpretive Threats

As in all Glaserian GT, coding, memoing, and theoretical integration rely on the researcher's judgment. We sought to cultivate theoretical sensitivity through sustained memoing, constant comparison across incidents, and disciplined theoretical coding to raise relationships to a conceptual level. Still, selective perception and premature closure constitute threats: researchers may inadvertently privilege familiar patterns or canonical SE narratives (e.g., "traceability always improves coupling management") over emergent properties.

Mitigation and residual risk: We maintained a memo bank, an audit of decision rules (e.g., when a property becomes dimensional), and we delayed the literature integration to the extent feasible in a secondary-study design. We also challenged emerging patterns via negative case analysis within the corpus. However, classic GT values emergent objectivity over inter-coder agreement; hence, some subjectivity remains inherent to the method and may be reflected in the delineation of categories and bridging relations.

5.5. Reliability, Auditability, and Reproducibility

In classic GT, reliability is not equated with inter-coder reliability; instead, emphasis is placed on a transparent analytic process that supports modifiability. We provide a documented audit trail (GSMS protocol, inclusion/exclusion rationales, code evolution, memo excerpts, theory integration decisions). Nevertheless, because emergence is sensitive to the researcher's theoretical sensitivity and to where comparisons are drawn, replications may yield different but legitimate conceptualizations (especially alternative theoretical codes linking the same categories). This variability is expected in classic GT but may concern readers seeking procedural reproducibility.

Mitigation and residual risk: By making the decision log, memo bank, and coding artifacts available (where possible), we enhance auditability. Still, alternative analysts may produce different theoretical delimitations of bridges and their properties.

5.6. Software Engineering-Specific Threats

SE is heterogeneous across domains, team structures, and tooling ecosystems. Reports vary in artifact granularity (e.g., fine-grained commit traces vs. high-level process descriptions) and may employ domain-specific lexicons (e.g., safety cases, hazard logs, user stories, architectural viewpoints). Such heterogeneity threatens comparability during constant comparison and may obscure bridge manifestations (e.g., "requirements-test" bridges vs. "ops telemetry-feature toggles" bridges). Additionally, publication bias and reporting deficits (e.g., missing negative outcomes, limited detail on failed integrations) may underreport bridge breakdowns (e.g., brittle traceability, unaligned architectural and testing strategies).

Mitigation and residual risk: We triangulated across lifecycle stages (requirements, design, implementation, testing, operations) and roles (analysts, developers, testers, SRE/DevOps). Nonetheless, variations in method (agile vs. plan-driven), regulation (regulated vs. non-regulated), and scale (component vs. system-of-systems) remain underexplored, limiting transferability.

5.7. Future Work

To address the above threats, future research will:

Extend theoretical sampling beyond the initial GSMS corpus to incorporate industrial case studies across diverse domains and development processes, actively seeking variation and negative cases to test the boundaries of the bridge construct.

Pursue theoretical saturation by iteratively sampling where comparisons are most fruitful, guided by the evolving core category and theoretical codes.

Develop operational indicators for bridge types in SE artifacts (e.g., traceability matrices, dependency networks, architectural conformance analyses, CI/CD pipeline handoffs) to support mixed-method evaluation (qualitative grounding with quantitative patterning).

Embrace modifiability by revising properties and dimensions of bridge typologies as new field data vary the pattern, ensuring the theory continues to fit, work, and remain relevant for practitioners.

Document audit artifacts (decision rules, memos, selective/theoretical coding moves) to enhance process transparency and enable informed theoretical replication.

6. Conclusions

The Glaserian Systematic Mapping Study (GSMS) enables the identification of emerging factors that may not be well represented in the existing literature, offering innovative insights and practices grounded in qualitative evidence. The emerging theories and theoretical propositions developed through GSMS can be directly applied to improve software engineering practices and processes, providing a solid foundation for informed decision-making. By generating Grounded Theories, GSMS can significantly contribute to the education and training of software engineers, offering theoretical and practical frameworks based on observed reality.

The Software Development Code Group (SDCG) comprises emergent elements from analysis conducted using Grounded Theory (GT) and the GSMS methodology. These elements were explored across various dimensions, particularly agility, data heterogeneity, core GT principles, and the presence of conceptual or methodological bridges across software engineering practices.

A high proportion of the SDCG components are directly associated with Agile methodologies. Elements, such as the Agile Transformation Process (ATP), knowledge sharing, pair programming, and team dynamics, are tied to Agile environments, supported by multiple studies within the dataset. Others, such as architecture, model–metamodels–UML, and company-related themes, appear more frequently in plan-driven or Non-Agile contexts. This dual presence highlights the importance of categorizing development practices along dimensions such as {Agile/Non-Agile} and {Agile/Plan-driven}, which emerged as significant classifications in the second iteration of the study.

Many SDCG elements also exhibit strong connections to heterogeneous data sources. Items such as pair programming, tools, testing, and team structures are discussed in the context of data collected through diverse methods, including interviews, observation, documentation, and software artifacts. These sources vary in format and structure, supporting the theoretical development of the {heterogeneous data} category and highlighting the importance of integrating structured, semi-structured, and unstructured data in the analysis of sociotechnical systems.

Moreover, several SDCG items align directly with the Grounded Theory elements (GT_e) framework. Concepts such as requirements, software development processes, and model–metamodels (UMLs) frequently appear within the {Basic Coding} and {Master Core Category} phases, supporting their theoretical relevance. In particular, pair programming stands out as a dominant theme in {A GT of ...} theories, indicating its central role in empirical GT-based studies.

One of the most insightful contributions of this categorization is the identification of conceptual {bridges}. These bridges represent cross-cutting interactions between technical and human-centered practices, connecting divergent methodologies, participant roles, or data paradigms. For instance, bridges emerge between Agile and plan-driven development, between theoretical models and industrial application, and between developers and other actors such as designers or users. Tools and testing practices, often associated with both Agile and Non-Agile contexts, further exemplify how certain elements mediate between contrasting domains. Similarly, team structures and open-source processes reflect bridges between individual and collective practices, particularly in distributed or collaborative environments.

The classification outlined in the table synthesizes the multidimensional nature of software engineering through a grounded theoretical lens. It demonstrates how methodological elements, sociotechnical structures, and practical applications are interrelated. Ultimately, the integration of GT into software engineering not only enables a richer categorization of development artifacts and practices but also exposes underlying theoretical patterns and cross-domain linkages that traditional taxonomies, such as those based solely on technical domains, fail to capture.

The data that GT accesses primarily pertain to social studies as they relate to human beings. In software development, these studies encompass the environment and reality of software participants and practitioners, as well as their artifacts. It results in a diverse collection of data, known as heterogeneous data.

GTe, as a general classification approach for GT across articles, regardless of the specific GT employed, has facilitated a broader, more in-depth categorization of software engineering articles.

Something that has enabled the identification of GT elements is the establishment of a diversity of interrelationships, which initially seemed to be isolated cases of GT use in software engineering and later proved to be more profound and more widespread. These interrelationships are observed as a whole, giving rise to a pattern in which GT has created bridges. The interrelationships form a pattern of bridges designed by GT among diverse elements of software development, such as actors, specialists, graphic designers, and participants, between Agile and plan-driven approaches, and across different stages of software development.

An additional bridge is the so-called multidimensional bridge, which, due to the iterative nature of the GT methodology, enables researchers to shift their focus or the dimension studied, yielding new, more complete, and more profound findings within the scope of their research.

GTe and its findings, such as heterogeneous data, bridges, and single- and multidimensional bridging, have enabled a way of classifying software engineering that differs from the traditional SWEBOK approach.

When working with GT in software engineering, it has been proven that GT is not limited to interviews, observations, surveys, focus groups, and social studies. It can also be applied to research results, engineering artifacts, and documents in general, allowing the development of strategies and conclusions using GT in the form of statements or expressions of emerging theories.

Although saturation is a methodological construct within Grounded Theory, its extension to software development is employed here strictly as a heuristic analogy. The term “engineering saturation” is not proposed as a formal property of software artifacts, but as a descriptive metaphor indicating perceived structural coherence within a given development context.

The concept of the *Bridge* encapsulates Grounded Theory’s integrative capacity to connect paradigms, methodologies, human factors, and practical applications in software

engineering. The presence of bridges across methodological, epistemological, and empirical domains reinforces GT's role as a unifying framework capable of managing complexity, fostering theoretical Synthesis, and promoting methodological innovation. These findings provide an articulated response to the research questions posed in this study and highlight the enduring value of GT as a means of reconciling diverse perspectives in the empirical analysis of software development.

Author Contributions: Conceptualization, G.N.; Investigation, G.N.; Methodology, G.N. and A.Y.; Writing, G.N.; Writing—Review and Editing, G.N. and A.Y.; Visualization, G.N.; Supervision, G.N. and A.Y.; Project administration, G.N. and A.Y.; All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Universidad Politécnica Salesiana and the IDEIAGEOCA Research Group under the project “Study and Synergy of Large Language Models: An Experimental Framework for Data Science and Software Engineering.”

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data supporting the results and analysis presented in this study can be accessed at <https://data.mendeley.com/datasets/7cvpkfydjv/1>, accessed on 10 February 2026.

Acknowledgments: We want to express our gratitude to “Doctorado en Software, Sistemas y Computación (DSSC)”, de la Escuela Técnica Superior de Ingenieros Informáticos, of the Polytechnic University of Madrid (UPM). Additionally, we would like to thank the members of the IDEIAGEOCA research group at the Universidad Politécnica Salesiana, as well as Elena Reascos, Gabriel Navas-Reascos, and Gustavo Navas-Reascos, who provided invaluable assistance throughout this investigation.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Adolph, S.; Hall, W.; Kruchten, P. A Methodological Leg to Stand on: Lessons Learned Using Grounded Theory to Study Software Development. In *Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds (CASCON '08)*, Richmond Hill, ON, Canada, 27–30 October 2008; ACM: New York, NY, USA, 2008; pp. 13:166–13:178. <https://doi.org/10.1145/1463788.1463806>.
2. Glaser, B. G.; Strauss, A. L., *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine, 1973.
3. Charmaz, K. *Constructing Grounded Theory. A Practical Guide Through Qualitative Analysis*. vol. 1. 2006. doi: 10.1017/CBO9781107415324.004.
4. Urquhart, C. An encounter with grounded theory: Tackling the practical and philosophical issues, in *Qualitative research in IS: Issues and trends*, IGI Global, 2001, pp. 104–140.
5. Matavire, R.; Brown, I. Profiling grounded theory approaches in information systems research, *European Journal of Information Systems*, vol. 22, no. 1, pp. 119–129, 2013.
6. Birks, D.F.; Fernandez, W.; Levina, N.; Nasirin, S. Grounded theory method in information systems research: its nature, diversity and opportunities, *European Journal of Information Systems*, vol. 22, no. 1, pp. 1–8, 2013.
7. Van Niekerk J.C.; Roode, J. Glaserian and Straussian Grounded Theory: Similar or Completely Different? in *Proceedings of the 2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists*, 2009, pp. 96–103. doi: 10.1145/1632149.1632163.
8. Bourque, P.; Fairley, R.E. *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*, 3rd ed.; IEEE Computer Society Press: Los Alamitos, CA, USA, 2014.
9. Navas, G.; Yagüe, A. Glaserian Systematic Mapping Study: An Integrating Methodology. In *Proceedings of the 17th International Conference on Evaluation of Novel Approaches to Software Engineering*, Online, 25–26 April 2022; SCITEPRESS Science and Technology Publications: Setubal, Portugal, 2022; pp. 519–527. <https://doi.org/10.5220/0011090500003176>.

10. Navas, G.; Yagüe, A. A New Way of Cataloging Research through Grounded Theory. *Appl. Sci.* **2023**, *13*, 10. <https://doi.org/10.3390/app13105889>.
11. Adolph, S.; Hall, W.; Kruchten, P. Using grounded theory to study the experience of software development. *Empir. Softw. Eng.* **2011**, *16*, 487–513. <https://doi.org/10.1007/s10664-010-9152-6>.
12. Siau, K.; Tian, Y. Open-Source Software Development Process Model. *J. Glob. Inf. Manag.* **2013**, *21*, 103–120. <https://doi.org/10.4018/jgim.2013100106>.
13. Haenni, N.; Lungu, M.; Schwarz, N.; Nierstrasz, O. Categorizing developer information needs in software ecosystems. In Proceedings of the 2013 International Workshop on Ecosystem Architectures, Saint Peterburg, Russia, 19 August 2013; pp. 1–5. <https://doi.org/10.1145/2501585.2501586>.
14. Zayour, I.; Hamdar, A. A qualitative study on debugging under an enterprise IDE. *Inf. Softw. Technol.* **2016**, *70*, 130–139. <https://doi.org/10.1016/j.infsof.2015.10.010>.
15. Di Penta, M.; Tamburri, D.A. Combining quantitative and qualitative studies in empirical software engineering research. In Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering Companion, ICSE-C 2017, Buenos Aires Argentina, 20–28 May 2017; pp. 499–500. <https://doi.org/10.1109/ICSE-C.2017.163>.
16. Tamburri, D.A.; Kazman, R. General methods for software architecture recovery: A potential approach and its evaluation. *Empir. Softw. Eng.* **2018**, *23*, 1457–1489. <https://doi.org/10.1007/s10664-017-9543-z>.
17. Ibrahim, M.H.; Darwish, N.R. Investigation of Adherence Degree of Agile Requirements Engineering Practices in Non-Agile Software Development Organizations. *Int. J. Adv. Comput. Sci. Appl.* **2015**, *6*, 97–103.
18. Sharp, J.H.; Mitchell, A.; Lang, G. Agile Teaching and Learning in Information Systems Education: An Analysis and Categorization of Literature. *J. Inf. Syst. Educ.* **2020**, *31*, 269–281.
19. Tessem, B. Individual empowerment of agile and non-agile software developers in small teams. *Inf. Softw. Technol.* **2014**, *56*, 873–889. <https://doi.org/10.1016/j.infsof.2014.02.005>.
20. Melnik, G.; Maurer, F. Comparative analysis of job satisfaction in agile and non-agile software development teams. In *Extreme Programming and Agile Processes in Software Engineering*; Abrahamsson, P., Marchesi, M., Succi, G., Eds.; Lecture Notes in Computer Science; Heidelberg Platz 3, D-14197; Springer: Berlin, Germany, 2006; Volume 4044, pp. 32–42.
21. Morse, J.M.; Stern, P.N.; Corbin, J.; Bowers, B.; Charmaz, K.; Clarke, A. *Grounded Theory: The Second Generation*; Left Coast Press: Walnut Creek, CA, USA, 2009. <https://doi.org/10.4324/9781315430577>.
22. Stol, K.-J.; Ralph, P.; Fitzgerald, B. Grounded theory in software engineering research: A Critical Review and Guidelines. In Proceedings of the 38th International Conference on Software Engineering—ICSE '16, Austin, TX, USA, 14–22 May 2016; pp. 120–131. <https://doi.org/10.1145/2884781.2884833>.
23. Hansen, B.H.; Kautz, K. Grounded Theory Applied—Studying Information Systems Development Methodologies in Practice. In Proceedings of the 38th Annual Hawaii International Conference on System Sciences, Big Island, HI, USA, 6 January 2005, p. 264b. <https://doi.org/10.1109/HICSS.2005.289>.
24. Treccani, P.J.F.; De Souza, C.R.B. Collaborative refactoring: Results of an empirical study using grounded theory. In *Lecture Notes in Computer Science*; Including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics; CRIWG 2011: Paraty, Brazil, 2011; pp. 73–80. https://doi.org/10.1007/978-3-642-23801-7_6.
25. Wang, L. Heterogeneous Data and Big Data Analytics. *Autom. Control. Inf. Sci.* **2017**, *3*, 8–15. <https://doi.org/10.12691/acis-3-1-3>.
26. Google Cloud. Cloud Natural Language. Available online: <https://cloud.google.com/natural-language/?hl=es> (accessed on 26 January 2019).
27. Würfel, D.; Lutz, R.; Diehl, S. Grounded requirements engineering: An approach to use case driven requirements engineering. *J. Syst. Softw.* **2016**, *117*, 645–657. <https://doi.org/10.1016/j.jss.2015.10.024>.
28. Ardo, A.A.; Bass, J.M.; Gaber, T. Towards Secure Agile Software Development Process: A Practice-Based Model. In Proceedings of the 2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Gran Canaria, Spain, 31 August–2 September 2022; pp. 149–156. <https://doi.org/10.1109/SEAA56994.2022.00031>.
29. Pina, D.; Seaman, C.; Goldman, A. Technical Debt Prioritization: A Developer’s Perspective. In Proceedings of the International Conference on Technical Debt (TechDebt 2022), Pittsburgh, PA, USA, 22–23 May 2022; pp. 46–55. <https://doi.org/10.1145/3524843.3528096>.
30. Jarzębowicz, A.; Weichbroth, P. A Qualitative Study on Non-Functional Requirements in Agile Software Development. *IEEE Access* **2021**, *9*, 40458–40475. <https://doi.org/10.1109/ACCESS.2021.3064424>.

31. De Souza Santos, R.E.; Ralph, P. A Grounded Theory of Coordination in Remote-First and Hybrid Software Teams. In Proceedings of the International Conference on Software Engineering, Pittsburgh, PA, USA, 19–20 May 2022; pp. 25–35. <https://doi.org/10.1145/3510003.3510105>.
32. Pillay, N.; Wing, J. Agile UX: Integrating good UX development practices in Agile. In Proceedings of the 2019 Conference on Information Communications Technology and Society (ICTAS), Durban, South Africa, 6–8 March 2019; pp. 2–7. <https://doi.org/10.1109/ICTAS.2019.8703607>.
33. Farias, R.S.; de Souza, R.M.; McGregor, J.D.; de Almeida, E.S. Designing smart city mobile applications: An initial grounded theory. *Empir. Softw. Eng.* **2019**, *24*, 3255–3289. <https://doi.org/10.1007/s10664-019-09723-8>.
34. Lubin, J. How Statically-Typed Functional Programmers Author Code. In *Proceedings of the Conference on Human Factors in Computing Systems, Yokohama, Japan, 8–13 May 2021*; Association for Computing Machinery: New York, NY, USA, 2021. <https://doi.org/10.1145/3411763.3451515>.
35. Danilova, A.; Naiakshina, A.; Smith, M. One size does not fit all: A grounded theory and online survey study of developer preferences for security warning types. In Proceedings of the International Conference on Software Engineering, Seoul, Republic of Korea, 27 June–19 July 2020; pp. 136–148. <https://doi.org/10.1145/3377811.3380387>.
36. Moshtari, S.; Okutan, A.; Mirakhorli, M. A Grounded Theory Based Approach to Characterize Software Attack Surfaces. In Proceedings of the 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE), Pittsburgh, PA, USA, 25–27 May 2022. <https://doi.org/10.1145/3510003.3510210>.
37. Seth, F.P.; Mustonen-Ollila, E.; Taipale, O.; Smolander, K. Software quality construction in 11 companies: An empirical study using the grounded theory. *Softw. Qual. J.* **2015**, *23*, 627–660. <https://doi.org/10.1007/s11219-014-9246-2>.
38. Baskerville, R.; Pries-Heje, J.; Madsen, S. Post-agility: What follows a decade of agility? *Inf. Softw. Technol.* **2011**, *53*, 543–555. <https://doi.org/10.1016/j.infsof.2010.10.010>.
39. Martakis, A.; Daneva, M. Handling requirements dependencies in agile projects: A focus group with agile software development practitioners. In Proceedings of the International Conference on Research Challenges in Information Science, Paris, France, 29–31 May 2013. <https://doi.org/10.1109/RCIS.2013.6577679>.
40. Kasurinen, J.; Taipale, O.; Smolander, K. Test case selection and prioritization. In Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement—ESEM '10, Bolzano-Bozen, Italy, 16–17 September 2010; p. 1. <https://doi.org/10.1145/1852786.1852800>.
41. Kettunen, V.; Kasurinen, J.; Taipale, O.; Smolander, K. A study on agility and testing processes in software organizations. In Proceedings of the 19th International Symposium on Software Testing and Analysis—ISSTA '10, Trento, Italy, 12–16 July 2010; pp. 231–240. <https://doi.org/10.1145/1831708.1831737>.
42. Kaskenpalo, P.; MacDonell, S.G. Valuing evaluation: Methodologies to bridge research and practice. In Proceedings of the 2nd International Workshop on Evidential Assessment of Software Technologies (EAST'12), Lund, Sweden, 22 September 2012. <https://doi.org/10.1145/2372233.2372242>.
43. Fernández, D.M.; Wagner, S. Naming the pain in requirements engineering: A design for a global family of surveys and first results from Germany. *Inf. Softw. Technol.* **2015**, *57*, 616–643. <https://doi.org/10.1016/j.infsof.2014.05.008>.
44. Rafi, S.; Yu, W.; Akbar, M.A. Towards a Hypothetical Framework to Secure DevOps Adoption: Grounded Theory Approach. In Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering (EASE'20), Trondheim, Norway, 15–17 April 2020; pp. 457–462. <https://doi.org/10.1145/3383219.3383285>.
45. Tuape, M.; Hasheela-Mufeti, V.T.; Kasurinen, J. Theory on Non-Technical Characteristics Affecting Process Adoption in Small Software Companies: A Grounded Theory Study. *IEEE Access* **2022**, *10*, 103382–103400. <https://doi.org/10.1109/ACCESS.2022.3209673>.
46. Brown, J.; Lindgaard, G.; Biddle, R. Interactional identity: Designers and developers making joint work meaningful and effective. In Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work (CSCW '12), Washington, DC, USA, 11–15 February 2012; pp. 1381–1390. <https://doi.org/10.1145/2145204.2145409>.
47. Turner, G.; Weakley, A.; Zhang, Y.; Edmonds, E. Attuning: A Social and Technical Study of Artist–Programmer Collaborations. *Proc. PPiG* **2005**, *17*, 106–119.
48. Lutz, R.; Wurfel, D.; Diehl, S. How Humans Merge UML-Models. In Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement, Edmonton, AB, Canada, 22–23 September 2011; pp. 177–186. <https://doi.org/10.1109/ESEM.2011.26>.

49. Salinger, S.; Zieris, F.; Prechelt, L. Liberating pair programming research from the oppressive driver/observer regime. In Proceedings of the International Conference on Software Engineering, San Francisco, CA, USA, 18–26 May 2013; pp. 1201–1204. <https://doi.org/10.1109/ICSE.2013.6606678>.
50. Bass, J.M. How product owner teams scale agile methods to large distributed enterprises. *Empir. Softw. Eng.* **2015**, *20*, 1525–1557. <https://doi.org/10.1007/s10664-014-9322-z>.
51. Last, M.Z. Understanding the group development process in global software teams. In Proceedings of the Frontiers in Education Conference (FIE), Westminster, CO, USA, 5–8 November 2003. <https://doi.org/10.1109/FIE.2003.1265928>.
52. Counsell, S.; Phalp, K.; Mendes, E.; Geddes, S. The Concerns of Prototypers and Their Mitigating Practices: An Industrial Case-Study. In Proceedings of the PROFES 2006, Amsterdam, The Netherlands, 12–14 June 2006, pp. 166–176. https://doi.org/10.1007/11767718_16.
53. Lim, N.K.; Ang, J.S.K.; Parvi, F.N. Diffusing software-based innovation with a software factory approach for software development. In Proceedings of the 2000 IEEE International Conference on Management of Innovation and Technology, Singapore, 12–15 November 2000; pp. 549–555. <https://doi.org/10.1109/ICMIT.2000.916749>.
54. Cheng, P.H.; Chang, H.C.; Chang, F.H. Another extensible requirements elicitation and analysis method. In Proceedings of the 2012 4th International Conference on Computational Intelligence, Communication Systems and Networks (CICSyN 2012), Washington, DC, USA, 3–5 November 2012; pp. 161–166. <https://doi.org/10.1109/CICSyN.2012.38>.
55. Badreddin, O.; Lethbridge, T.C. Combining experiments and grounded theory to evaluate a research prototype: Lessons from the umple model-oriented programming technology. In Proceedings of the 2012 1st International Workshop on User Evaluation for Software Engineering Researchers (USER 2012), Zurich, Switzerland, 5 June 2012; pp. 1–4. <https://doi.org/10.1109/USER.2012.6226575>.
56. Looso, S.; Börner, R.; Goeken, M. Using grounded theory for method engineering. In Proceedings of the International Conference on Research Challenges in Information Science, Basse-Terre, France, 19–21 May 2011. <https://doi.org/10.1109/RCIS.2011.6006826>.
57. Ferrari, R.; Noorwali, I.; Madhavji, N.H. Towards a theory on the interaction between requirements engineering and systems architecting. In Proceedings of the 5th International Workshop on Empirical Requirements Engineering (EmpiRE 2015), Washington, DC, USA, 24–August 2015; pp. 17–20. <https://doi.org/10.1109/EmpiRE.2015.7431302>.
58. Javdani Gandomani, T.; Ziaei Nafchi, M. An empirically developed framework for Agile transition and adoption: A Grounded Theory approach. *J. Syst. Softw.* **2015**, *107*, 204–219. <https://doi.org/10.1016/j.jss.2015.06.006>.
59. Gandomani, T.J.; Zulzalil, H.; Abdul Ghani, A.A.; Abu, A.B.; Parizi, R.M. The impact of inadequate and dysfunctional training on agile transformation process: A grounded theory study. *Inf. Softw. Technol.* **2015**, *57*, 295–309. <https://doi.org/10.1016/j.infsof.2014.05.011>.
60. Hoda, R.; Noble, J.; Marshall, S. The impact of inadequate customer collaboration on self-organizing Agile teams. *Inf. Softw. Technol.* **2011**, *53*, 521–534. <https://doi.org/10.1016/j.infsof.2010.10.009>.
61. Gandomani, T.J.; Zulzalil, H.; Nafchi, M.Z. Agile transformation: What is it about? In Proceedings of the 2014 8th Malaysian Software Engineering Conference (MySEC 2014), Langkawi, Malaysia, 23–24 September 2014; pp. 240–245. <https://doi.org/10.1109/MySec.2014.6986021>.
62. Gandomani, T.J.; Zulzalil, H.; Abdul Ghani, A.A.; Sultan, A.B.M.; Sharif, K.Y. How human aspects impress Agile software development transition and adoption. *Int. J. Softw. Eng. Its Appl.* **2014**, *8*, 129–148. <https://doi.org/10.14257/ijseia.2014.8.1.12>.
63. Hoda, R.; Noble, J. Becoming Agile: A Grounded Theory of Agile Transitions in Practice. In Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE 2017), Buenos Aires, Portugal, 20–28 May 2017; pp. 141–151. <https://doi.org/10.1109/ICSE.2017.21>.
64. Ghanbari, H. Seeking technical debt in critical software development projects: An exploratory field study. In Proceedings of the Annual Hawaii International Conference on System Sciences, Koloa, HI, USA, 5–8 January 2016, pp. 5407–5416. <https://doi.org/10.1109/HICSS.2016.668>.
65. Ramesh, B.; Pries-Heje, J.; Baskerville, R. Internet software engineering: A different class of processes. *Ann. Softw. Eng.* **2002**, *14*, 169–195. <https://doi.org/10.1023/A:1020557725165>.
66. Sanchez-Gordon, M.L.; O'Connor, R.V.; Colomo-Palacios, R. Evaluating VSEsviewpoint and sentiment towards the ISO/IEC 29110 standard: A two Country grounded theory study. In Proceedings of the Communications in Computer and Information Science, Proceedings of SPICE 2015, Gothenburg, Sweden, 16–17 June 2015; pp. 114–127. https://doi.org/10.1007/978-3-319-19860-6_10.

67. Rodríguez, P.; Urquhart, C.; Mendes, E. A Theory of Value for Value-Based Feature Selection in Software Engineering. *IEEE Trans. Softw. Eng.* **2022**, *48*, 466–484. <https://doi.org/10.1109/TSE.2020.2989666>.
68. MacCarthy, R.W.; Bass, J.M. The Role of Skillset in the Determination of DevOps implementation Strategy. In Proceedings of the 2021 IEEE/ACM Joint 15th International Conference on Software and System Processes and 16th ACM/IEEE International Conference on Global Software Engineering (ICSSP/ICGSE 2021), Virtual, 18–19 May 2021; pp. 50–60. <https://doi.org/10.1109/ICSSP-ICGSE52873.2021.00015>.
69. Salman, I.; Rodríguez, P.; Turhan, B.; Tosun, A.; Gureller, A. What Leads to a Confirmatory or Disconfirmatory Behavior of Software Testers? *IEEE Trans. Softw. Eng.* **2022**, *48*, 1351–1368. <https://doi.org/10.1109/TSE.2020.3019892>.
70. Dissanayake, N.; Zahedi, M.; Jayatilaka, A.; Babar, M.A. A Grounded Theory of the Role of Coordination in Software Security Patch Management. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2021)*; Association for Computing Machinery: New York, NY, USA, 2021; pp. 793–805. <https://doi.org/10.1145/3468264.3468595>.
71. Dantas, E.; Neto, A.S.; Valadares, D.; Perkusich, M.; Ramos, F.; Almeida, H.; Perkusich, A. Investigating technological risks and mitigation strategies in software projects. In Proceedings of the ACM Symposium on Applied Computing, Virtual, 25–29 April 2022; pp. 1527–1535. <https://doi.org/10.1145/3477314.3507062>.
72. Ayas, H.M.; Leitner, P.; Hebig, R. Facing the giant: A grounded theory study of decision-making in microservices migrations. In Proceedings of the International Symposium on Empirical Software Engineering and Measurement, Bari, Italy, 11–15 October 2021. <https://doi.org/10.1145/3475716.3475792>.
73. Hoda, R.; Noble, J.; Marshall, S. Organizing self-organizing teams. In Proceedings of the 32nd International Conference on Software Engineering, Cape Town, South Africa, 1–8 May 2010.
74. Masood, Z.; Hoda, R.; Blincoe, K. How agile teams make self-assignment work: A grounded theory study. *Empir. Softw. Eng.* **2020**, *25*, 4962–5005. <https://doi.org/10.1007/s10664-020-09876-x>.
75. Weimann, P.; Pollock, M.; Scott, E.; Brown, I. Enhancing team performance through tool use: How critical technology-related issues influence the performance of virtual project teams. *IEEE Trans. Prof. Commun.* **2013**, *56*, 332–353. <https://doi.org/10.1109/TPC.2013.2287571>.
76. Santos, V.; Goldman, A.; de Souza, C.R.B. Fostering effective inter-team knowledge sharing in agile software development. *Empir. Softw. Eng.* **2015**, *20*, 1006–1051. <https://doi.org/10.1007/s10664-014-9307-y>.
77. Whitworth, E.; Biddle, R. Motivation and Cohesion in Agile Teams. In *Agile Processes in Software Engineering and Extreme Programming. In Proceedings of the XP 2007, Como, Italy, 18–22 June 2007*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2007; pp. 62–69. https://doi.org/10.1007/978-3-540-73101-6_9.
78. Stojanov, Z.; Dobrilovic, D.; Jevtic, V. Identifying properties of software change request process: Qualitative investigation in very small software companies. In Proceedings of the 9th International Symposium on Intelligent Systems and Informatics, Proceedings (SISY 2011), Subotica, Serbia, 8–10 September 2011; pp. 47–52. <https://doi.org/10.1109/SISY.2011.6034369>.
79. Al-Ani, B.; Marczak, S.; Redmiles, D.; Prikladnicki, R. Facilitating contagion trust through tools in Global Systems Engineering teams. *Inf. Softw. Technol.* **2014**, *56*, 309–320. <https://doi.org/10.1016/j.infsof.2013.11.001>.
80. Chitchyan, R.; Bird, C. Theory as a Source of Software Requirements. In *Proceedings of the IEEE International Conference on Requirements Engineering, Zurich, Switzerland, 31 August–4 September 2020*; IEEE Computer Society: Los Alamitos, CA, USA 2020, pp. 227–237. <https://doi.org/10.1109/RE48521.2020.00033>.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.