Defining Key Concepts in Information Science Research: The Adoption of the Definition of Feature

Sabine Molenaar, Emilie Steenvoorden, Nikita van den Berg, Fabiano Dalpiaz[0000-0003-4480-3887], and Sjaak Brinkkemper[0000-0002-2977-8911]

Dept. of Information and Computing Sciences, Utrecht University, The Netherlands
{s.molenaar, e.r.m.steenvoorden, i.a.n.vandenberg, f.dalpiaz, s.brinkkemper}@uu.nl

Abstract. This paper analyzes the definitions of the concept *feature* in the information science literature. The concept of feature has been defined in various ways over the last three decades. To be able to obtain a common understanding of a feature in information science, it is important to conduct a thorough analysis of the definitions that can be used in research and in practice. The main contribution of this paper is a categorization of the existing definitions, which highlights similarities and differences. By means of a *Concept Definition Review* process, we gather a total of 23 definitions from Google Scholar using five search queries complemented by backward snowballing. Our analysis organizes the definitions according to their level of abstraction and the taken viewpoint. Within the range of analyzed definitions, we do not wish to argue that one is better or worse than another. We provide, however, guidelines for the selection of a definition for a given goal. These guidelines include: popularity based on the citations count, the research field, the abstraction level, and the viewpoint.

Keywords: Feature · Requirements Engineering · Information Science · Definition · Literature Review · Concept Definition Review

1 Introduction

The concept of *feature*, in relation to software and information systems, has been defined in many ways over the last three decades. One of the first definitions dates back to the 1990s, and it is stated in a highly influential technical report on feature-oriented domain analysis [20]. This definition seems to be adapted from the American Heritage dictionary entry for *feature*. Ever since, alternative definitions of the concept of *feature* emerged, which deviated from it.

The existence of multiple, diverging definitions has both conceptual and practical consequences. Conceptually, researchers may use the same terminology while referring to different meanings (*denotations* [28]), leading to undetected conflicts in verbal or written communication. Practically, the choice of a definition may affect the artifacts that are created based on the concept. For instance, since features are at the basis of feature diagrams [20], different definitions may lead to conflicting interpretations of a feature diagram, or to different models for the same system depending on the modeler's preferred definition.

To reach the goal of analyzing the different definition of feature, we sketch a more general literature review approach that we call *Concept Definition Review (CDR)*. Literature reviews are a widely practiced type of research method in various flavors in all scientific disciplines [12]: systematic literature review, meta-analysis, argumentative literature review, systematic mapping review. This new approach was made necessary by the need to identify the definitions of a certain term without conducting a heavyweight systematic literature review. Definitions of newly introduced concepts are usually made in an explicit statement (e.g., "We define the concept of *feature* as follows ...") at the beginning of the paper in order to establish a common understanding with the reader. For the CDR, the definition text with some context suffices and the remainder of the paper is then ignored. We envision that CDRs can be used to bring clarity regarding several concepts in the domains of information science, information systems, and software engineering, e.g., those of class, function, task, and goal.

We choose to investigate the concept of feature because of its importance both in Requirements Engineering (RE) and in Software Architecture (SA). For example, in our RE4SA framework [30], features are an elementary abstraction to define the functional architecture of a system, and atomic functional requirements are expected to justify individual features.

The rest of the paper is organized as follows. In Section 2, we describe the Concept Definition Review approach and its application to the concept of feature. In Section 3, we analyze the types of definitions, the research topics, the abstraction level, and the viewpoint on the concept of feature. Finally, we provide guidelines for researchers on the usage of the definitions of feature, and we conclude, in Section 4.

2 Research Method and Data Collection

The goal of this research is to recommend definitions that fit various perspectives and multiple purposes, rather than that of creating an exhaustive list of all definitions of the concept of feature. We are particularly interested in collecting and analyzing definitions from the domains of RE and SA. Therefore the main research question for this paper is formulated as follows: *"How are features defined for different purposes in the context of information science literature?"*.

2.1 Concept Definition Review

In an attempt to properly define the concept feature and categorize existing definitions in the field of information science, we devised a literature research method that focuses on analyzing and clarifying the meaning of a concept in the literature. The high-level structure of the CDR method consists of the following six steps, which are inspired by the SLR guidelines by Okoli and Schabram [25]:

1. *Purpose of the concept definition review:* the goal and research scope of the concept at hand is established by selecting the scientific sub-domains where the concept plays a critical role;
2. *Searching for papers containing definitions:* querying bibliographic indexes with the name of the concept in the identified scientific sub-domains;
3. *Relevancy screen and quality appraisal:* for each paper that fulfills the quality criteria of renowned scientific publication venues, an explicit formulation of the concept definition is to be identified;
4. *Data extraction:* various data items are collected from the literature resources, e.g. syntactical structure, research domain, and citation impact data;
5. *Synthesis of studies:* analysis of the data items provides insights on concept definition adoption, variations over abstraction levels, and evolution over time, i.e. old-fashioned definitions versus new interpretations (see, for instance, the changes of definitions by the same author group in Table 3);
6. *Documenting the concept guidelines:* based on the synthesis findings, a guideline is formulated for the most suitable concept definition usage in the research domains.

The remainder of this paper describes an instantiation of the CDR for the concept of *feature*. The six steps are illustrated by presenting our experience. A more general definition of the techniques that can be used is left to future work.

In our research, the definitions are collected by searching in Google Scholar, and by complementing those identified sources via backward snowballing. After the definitions are identified, we gathered the number of citations. The data collection is done at two points in time to observe usage evolution. The collection started in November 2018, then the research paused for two years, and additional data is collected in October 2021. The collected information is then analyzed from various perspectives. We first analyze the definition type and the research topic based on the abstract, keywords, introduction, and research topic of the venue where it has been published. Next, a categorization is made based on the level of abstraction and the viewpoint. The differentiation between abstract and technical is done through the method proposed by Apel and Kästner [4]. Based on the data collection and the analysis, we provide guidelines on how to select a suitable definition of *feature* for use in a given context.

2.2 Data collection

Initially, relevant papers are found using the search operators in Google Scholar. Through citation tracing, other literature repositories became involved (Scopus, ACM DL, IEEE Xplore, etc.). All definitions should be related to the term *feature*; thus, this term is included in all search queries. Since that term in combination with the term 'definition' often leads to results not related to information science, more specific queries were used instead. The second term in the search query is based on other topics relevant to this paper, as explained in the introduction: RE and SA. This is aligned with the objective of our research, which aims at identifying definitions for various purposes, but within the research sub-fields we have defined.

To scale down the number of results and to assure quality, some results are excluded. The definition and the source where the definition appears should meet the following selection criteria:

– They must be written in English.
– They must be scientific literature.
– They must present a unique definition of the concept feature.

Table 1 provides an overview of the used search queries and their included results. At first, we hoped we could restrict our search to a limited number of citations. However, since the works used in the literature study range from 1990 to 2021, this would be an unfair criterion, since older works have had more time to get cited. The third criterion traces to the original first definition of the concept. Furthermore, since the aim is to provide an overview of existing definitions, less cited definitions should be featured as well for completeness. The results are presented in order of the search results (relevance in relation to the search query). It should be noted that the work by Apel et al., published in 2013, is stated as a work from 2016 by Google Scholar. However, the book itself includes a copyright text from 2013 and the foreword was also dated 2013. The identified results that did not match the selection criteria are not listed in the table.

Table 1. Overview of the search queries on Google Scholar and of the returned results.

Search query	**Included results**
"feature" AND "requirements engineering"	Classen et al., 2008; Kang et al., 1990
"feature" AND "software architecture"	Apel & Kästner, 2009; Kang et al., 1990; Zhang et al., 2019
"feature" AND "product lines"	Apel et al., 2013
"feature" AND "software system"	Apel et al., 2013; Apel & Kästner, 2009
"feature" AND "feature-oriented specification"	Guerra et al., 1996; Apel & Kästner, 2009
"feature" AND "source code"	Dit et al., 2013

In addition, the snowballing technique was utilized. In this case, this consisted of backwards searching. Two articles were selected as a starting point, since these two works explicitly cited various definitions of the term *feature*. Table 2 summarizes which and how many works have been found per article.

Table 2. Works found through the use of the backward snowballing technique.

Source	**References**	***Total***
Classen et al., 2008	Kang et al., 1990; Kang et al., 1998; Bosch, 2000; Czarnecki & Eisenecker, 2000; Batory, 2004; Batory et al., 2004; Pohl et al., 2005; Batory et al., 2006; Apel et al., 2007	*9*
Apel & Kästner, 2009	Kang et al., 1990; Kang et al., 1998; Bosch, 2000; Czarnecki & Eisenecker, 2000; Zave, 2003; Batory et al., 2004; Chen et al., 2005; Czarnecki et al., 2005; Pohl et al., 2005; Batory et al., 2006; Apel et al., 2007; Classen et al., 2008; Kästner et al., 2008	*13*

Table 3. Definitions of *feature* obtained in our instantiation of the CDR. The 'Year' column refers to the year of publication.

Authors	**Year**	**Definition**
Kang, Cohen, Hess, Novak & Peterson [20]	1990	*"a prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems"*
Guerra, Ryan & Sernadas [17]	1996	*"is a part or aspect of a specification which a user perceives as having a self-contained functional role"*
Kang, Kim, Lee, Kim, Shin & Huh [19]	1998	*"distinctively identifiable functional abstractions that must be implemented, tested, delivered, and maintained"*
Bosch [9]	2000	*"a logical unit of behaviour specified by a set of functional and non-functional requirements"*
Czarnecki & Eisenecker [13]	2000	*"a distinguishable characteristic of a concept (e.g., system, component, and so on) that is relevant to some stakeholder of the concept"*
Zave [33]	2003	*"an optional or incremental unit of functionality"*
Batory [6]	2004	*"the primary units of software modularity"*
Batory, Sarvela & Rauschmayer [5]	2004	*"a product characteristic that is used in distinguishing programs within a family of related programs"*
Chen, Zhang, Zhao & Mei [10]	2005	*"a product characteristic from user or customer views, which essentially consists of a cohesive set of individual requirements"*
Czarnecki, Helsen & Eisenecker [14]	2005	*"a system property that is relevant to some stakeholder and is used to capture commonalities or discriminate among systems in a family"*
Pohl, Böckle & van der Linden [26]	2005	*"an end-user visible characteristic of a system"*
Batory, Benavides & Ruiz-Cortes [7]	2006	*"an increment in product functionality"*
Apel, Lengauer, Batory, Möller & Kästner [3]	2007	*"a structure that extends and modifies the structure of a given program in order to satisfy a stakeholder's requirement, to implement and encapsulate a design decision, and to offer a configuration option."*
Classen, Heymans & Schobbens [11]	2008	*"a triplet, f = (R,W,S), where R represents the requirements the feature satisfies, W the assumptions the feature takes about its environment and S its specification"*
Kästner, Apel & Kuhlemann [21]	2008	*"represents an increment in functionality relevant to stakeholders"*
Apel & Kästner [4]	2009	*"is a unit of functionality of a software system that satisfies a requirement, represents a design decision, and provides a potential configuration option"*
Apel, Batory, Kästner & Saake [2]	2013	*"is a characteristic or end-user-visible behavior of a software system"*
Dit, Revelle, Gethers & Poshyvanyk [15]	2013	*"represents a functionality that is defined by requirements and accessible to developers and users"*
Berger, Lettner, Rubin, Grünbacher, Silva, Becker, Chechik & Czarnecki [8]	2015	*"describe the functional and non-functional characteristics of a system"*
Andam, Burger, Berger & Chaudron [1]	2017	*"are high-level, domain-specific abstractions over implementation artifacts"*
Krüger, Gu, Shen, Mukelabai, Hebig & Berger [23]	2018	*"used to specify, manage, and communicate the behavior of software systems and to support developers in comprehending, reusing, or changing these systems"*
Rodríguez, Mendes & Turhan [27]	2018	*"represent needs that are gathered via meetings with customers or other stakeholders, which, once selected, are refined during a requirements elicitation process"*
Zhang, Wang & Xie [34]	2019	*"indispensably basic functional modules available to users, which can be captured by one or two words in the review"*

Overlapping references between the two works are included for both in the interest of completeness. Based on correspondence with Sven Apel, an additional three works co-written by Thorsten Berger are included (S. Apel, personal communication, February 12, 2019). More than one definition written by Apel is included and the article providing an overview of feature-oriented development is used not only as a starting point for searching for more definitions, but also because it inspired the synthesis in part. Therefore, the recommendation was gladly accepted. Moreover, these three works are published more recently than most of the other included works, providing a scientific evolution of the term *feature* over the past thirty years.

Table 3 shows the feature definitions in chronological order and, if two or more works were published in the same year, alphabetical order is applied. The references are provided via short citations to increase the readability of the table. The table highlights the high number of definitions of the term *feature* in the context of RE and SA: we identified 23 relevant ones. This leads to possible ambiguity and conflict [28] when discussing the literature in the field, and also when interpreting or creating models that refer to the concept of feature such as feature diagrams [20].

2.3 Popularity of the Definitions

After collecting the definitions, additional data about the number of citations per work was gathered. This data can suggest a first selection of a definition to use. Fig. 1 shows the number of citations per article measured in 2018 (blue) and 2021 (green). Also, the percentage growth of the number of citations in this time period is visible for each work. Interestingly, from the three most cited works, Kang et al., Eisenecker and Czarnecki and Pohl et al., the latter two experienced a stronger growth: 32% and 29%, respectively, much higher than the 5% growth of Kang et al. Since more recently published works have had less time to get cited, the picture may convey a slightly skewed view. Therefore, the publication year should also be taken into account when selecting the most cited works. This can be done by looking at a trend line. From the trend line based on the citations measured in 2021, it is apparent that four works are cited significantly more often relative to the others, being the works from Kang et al. (+271 citations), Eisenecker and Czarnecki (+1,177), Bosch (+279), and Dit et al. (+245). However, using only the number of citations does not take research topics into account.

3 Analysis and Categorization of the Concept Definitions

The previous section provided a basic recommendation for selecting a definition based on the number of citations. That straightforward criterion, which is very easy to adopt, does not actually answer the question as to what definition should be used in what context. To answer this question, we propose various analyses of the definitions of Table 3, which include the categorization of the definitions according to various facets: definition type, research topic, level of abstraction, and viewpoint.

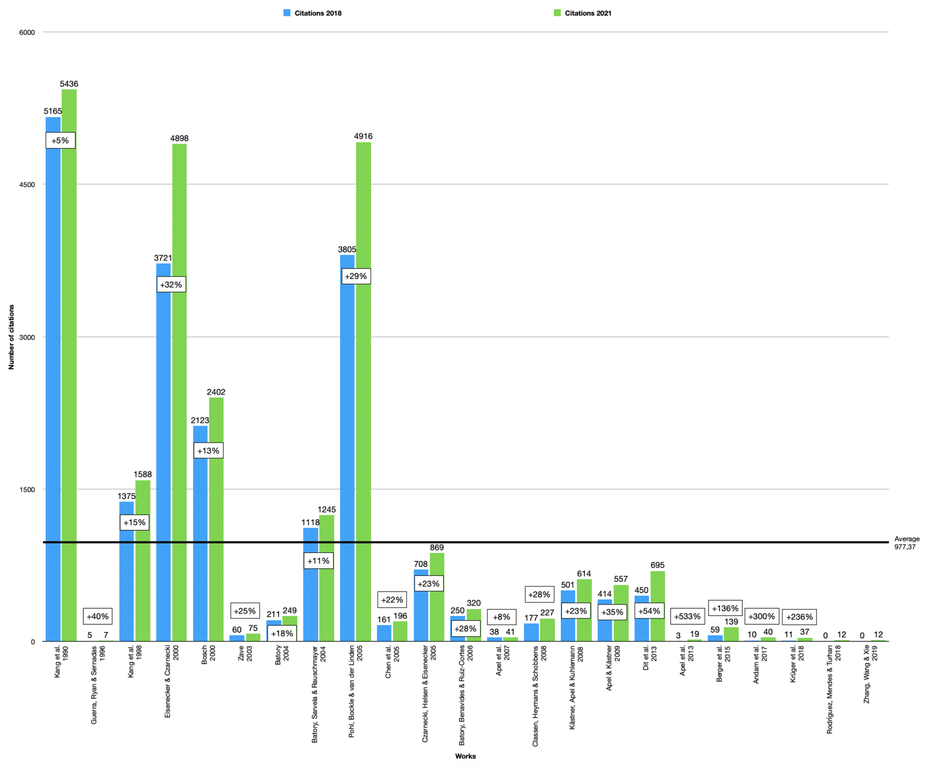


Fig. 1. Overview of number of citations on Google Scholar of works in which definitions are provided (from November 2018 to October 2021).

3.1 Definition types

Within the broad range of types of definitions, an important distinction is that between *intensional* and *extensional*.

An *intensional* definition describes the common characteristics of the members of the category, e.g., birds have feathers, they can fly, and they have a specific shape [16]. For example, Kang et al. (1990): *"a prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems"*. Here, a feature is described by means of common characteristics. Another example is that of Zhang et al. (2019): *"indispensably basic functional modules available to users, which can be captured by one or two words in the review"*. An *extensional* definition lists the members of the category, e.g. robins, eagles, nightingale, etc. Furthermore, other definition types exist. For example, *ostensive* definitions, which are like extensional definitions, but where extensional definitions call for an exhaustive list of members of the category, ostensive definitions only call for a couple of example members [32].

Besides this main distinction, other types of definitions can be identified. First off, a *stipulative* definition is used when a term is made up for the first time. Often consisting of a general category the concept belongs to, followed by a differentiator. Secondly, *lexical* definitions provide descriptions that depends on the term's use in particular communities: the definition depends on the audience. Common examples come from the legal domain. *Partitive* definitions explain concepts as being part of a greater whole. A partitive definition of feature is that of Guerra et al. (1996): *"is a part or aspect of a specification which a user perceives as having a self-contained functional role"*. Next, *functional* definitions explain actions or activities of a concept in relation to another concept. The definition of Batory et al. (2004) is functional: *"a product characteristic that is used in distinguishing programs within a family of related programs"*. *Encyclopedic* definitions often include additional information on the concept. Next, *theoretical* definitions attempt to add an argument for a concept and can be compared to scientific hypothesis. The last type of definition is the *synonym* definition, which describe a concept by mentioning a similar concept [31].

All definitions in this research, except for those of Guerra et al. (1996) and Batory et al. (2004), are intensional definitions, because the definitions analyze the concepts into constituent characteristics. It would be interesting to explore whether other types of definitions would be suitable for providing a clear, homogeneous characterization of the notion of feature.

3.2 Research topics

To recommend definitions based on context, it is useful to see in what research field or sub-field a definition is proposed. This organization based on research topic is provided in Table 4.

Table 4. Feature definitions categorized by research topic.

Research topic	**Related works**
Feature-oriented software	Kang et al., 1990, 1998; Batory, 2004; Apel et al., 2007; Apel & Kästner, 2009; Dit et al., 2013; Zhang et al., 2019
Feature-oriented specifications	Guerra et al., 1996; Zave, 2003
Generative programming	Czarnecki & Eisenecker, 2000
Software product lines	Bosch, 2000; Batory et al., 2004; Pohl et al., 2005; Kästner et al., 2008; Apel et al., 2013; Berger et al., 2015; Andam et al., 2017; Krüger et al., 2018
Feature modeling	Chen et al., 2005; Czarnecki et al., 2005; Batory et al., 2006
Requirements engineering	Classen et al., 2008
Release planning	Rodríguez et al., 2018

The research topics are determined based on which topics or fields in the abstract, keywords or introduction. In addition, we also considered the research fields or topics related to the journal or conference proceedings where the work was published. Some overlap between the topics is possible, since some works include a more specific topic or field than others. For example, in Table 4, feature-oriented software may also be interpreted as feature-oriented programming in some cases, but to keep it more generic, the former topic description is used instead. In addition, it is possible that a definition could fit more than one research topic, in such cases the most important or prominent one is selected. For instance, the definition by Rodríquez et al. could also fit the RE topic, but it is categorized as release planning, since this was the main topic of the work.

This categorization per research topic can be used to select a definition to utilize in the context of one of the research topics. For topics that have multiple fitting definitions, the additional factor of number of citations can suggest a preference. However, different and more extensive approaches for establishing the most suitable concept definition could be envisioned.

3.3 Abstraction level

Some definitions might be more suitable than others given a certain context or aim. In all definitions, two aspects can be distinguished: abstraction level and viewpoint. The former aspect was inspired by the differentiation between abstract and technical feature definitions as proposed by Apel and Kästner [4]. They also recognize that features have more than one use and describe the differentiation as follows:

1. Abstract: "*features are abstract concepts of the target domain, used to specify and distinguish software systems*" (problem space)
2. Technical: "*features must be implemented in order to satisfy requirements*" (solution space)

Czarnecki and Eisenecker have separated the problem space from the solution space, in which the former focuses on domain-specific abstractions and the latter on implementation-oriented abstractions [13]. Apel and Kästner use this distinction to further define abstract and technical, relating abstract definitions to the problem space and technical definitions to the solution space [4]. In addition to this distinction between abstract and technical definitions, they have provided a list of ten definitions (all of which are also included in Table 3) and ordered them from abstract to technical. However, they have not described how they decided on which definition is more technical than another. Moreover, they identified seven abstract definitions and only three technical ones. In short, while the line between abstract and technical is clear, the gap between the two is not and the reasoning behind the order within both distinctions is vague at best.

To clarify the interpretations of abstract and technical, Sven Apel was asked to comment on the paper. He stated that the first seven definitions "take a user-centric/problem-space-centric perspective", while the eighth definition is only formal from an RE perspective. The last two definitions focus on the implementation and are thus solution-space-specific. He continues by saying that, within these categorizations, the definitions are more or less sorted by date (S. Apel, personal communication, February 12, 2019). To conclude, this approach was quite informal and therefore difficult to replicate. Moreover, it still does not solve the mystery of which definition is more abstract or technical than another. A more formal categorization is needed to tackle these challenges.

In an attempt to recreate and extend such an order based on level of abstraction (from abstract to technical), nine characteristics were extracted from the collection of 19 definitions (the other four were added later due to additional communication and refreshing of the data in 2021). The following nine characteristics were extracted:

– Abstract: characteristic, distinct (or variations thereof), aspect, abstraction
– Technical: specification, functionality (or variations thereof), requirements, behavior, unit

The identified abstract characteristics are assigned a score of 1, the technical characteristics receive a 0, then we divide this value by the number of characteristics, leading to a score between 0 and 1. In this case, 1 is the most abstract and 0 the most technical (or least abstract). A test comparing the order based on these nine characteristics and resulting score and the order of seven definitions as presented by Apel and Kästner resulted in the following findings:

– Six out of seven definitions were ordered differently.
– Two definitions were shifted three positions.
– If the line between abstract and technical is placed at 0.5, one definition shifts from abstract to technical and one is shifted the other way around.

Due to the deviation from the original order and the sensitivity of the placement of the abstract/technical line, it is concluded this approach has certain drawbacks. A second attempt, adopting a different interpretive approach, yielded better results. After analyzing the different characteristics of abstract and technical as stated by Czarnecki and Eisenecker, and Apel and Kästner, as discussed earlier in this section, the following eight characteristics were identified:

– Abstract: problem space, description of requirements, description of intended behavior and characteristic/abstract/abstraction
– Technical: solution space, satisfaction of requirements, implementation of intended behavior and functionality

Using this approach, with the same method for calculating a score, the 19 definitions were ordered once again (the results are shown in Fig. 2) with the following results:

– If the line between abstract and technical is placed at 0.5, none of the definitions shift from abstract to technical or vice versa.
– The three technical definitions are in the same order.
– Out of the seven abstract definitions, only two are out of order (and the order among those two is the same as in the order by Apel and Kästner).

To summarize, out of the ten definitions, only two were out of order (and disregarding the other definitions, those two were in the correct order). Another advantage of this approach is that it is not based on terms/characteristics extracted from the definition, but on theoretical resources by Czarnecki and Eisenecker, and Apel and Kästner. Furthermore, it should not be forgotten that it is unclear whether the original order as devised by Apel and Kästner is on an ordinal scale. It is reasonable to assume so, since the definitions are numbered. However, the reasoning behind this specific order is not thoroughly explained, apart from the descriptions of abstract and technical as stated previously. The one fully unambiguous aspect is the distinction between the abstract and technical definitions, since this was explicitly mentioned.

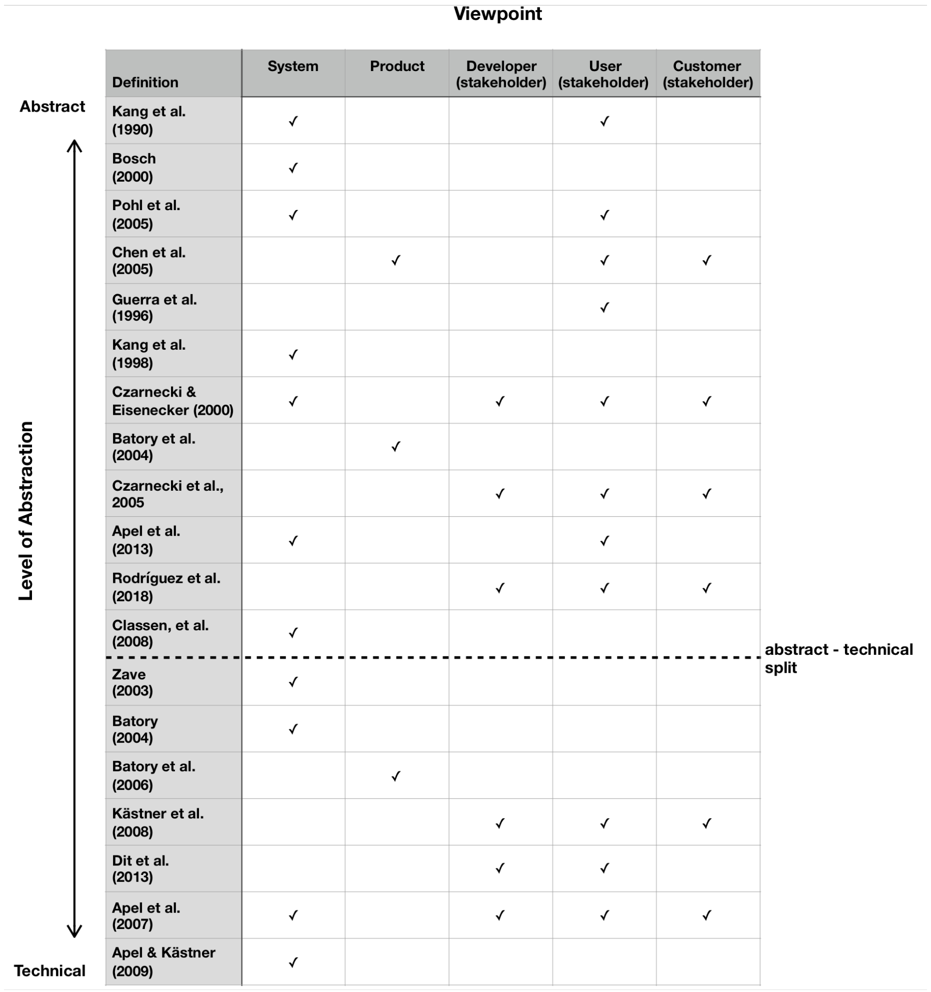
3.4 Viewpoint

In addition to the level of abstraction, five viewpoints were also extracted from exactly mentioned terms in the definitions:

– System
– Product
– Developer (stakeholder)
– User (stakeholder)
– Customer (stakeholder)

Firstly, system and product are considered separate viewpoints, since a system can be contained within a product, but a product can indicate more than just a system. Secondly, three stakeholders were identified and only human beings are considered a stakeholder. The developer was included, not because it was explicitly mentioned in any of the definitions, but sometimes the word stakeholder also refers to the development viewpoint. Thirdly, the user viewpoint also includes end-users and differs from the developer viewpoint, since developer do not necessarily use the product or system, but other employees of the product's or system's company might. Fourthly, customers are separated from user, since they are more specific than just any (end-) user. Finally, whenever no specific viewpoint is mentioned or can be reasonably assumed given a definition, the system is considered the viewpoint, due to features being part of the SA, which describes a system. Fig. 2 shows the categorization of the definitions based on the level of abstraction score (as described previously) and the identified viewpoints.

The 19 definitions and the scoring system were also presented to a group of 26 information science students and researchers. Both expressed a difficulty in understanding what the term 'technical' was supposed to mean in this context. Given their background, they automatically assumed technical characteristics to be related to development aspects or implementations (such as code). Moreover, the level of abstraction is often seen as the level of granularity, while in this categorization that is not the case. To make the categorization easier to read and understand, a different name and more specific minimal and maximum values would be desirable. Changing 'technical' to 'detailed' might solve the issue of


Viewpoint
Abstract
Level of Abstraction
Technical
abstract - technical split

Definition	System	Product	Developer (stakeholder)	User (stakeholder)	Customer (stakeholder)
Kang et al. (1990)	✓			✓	
Bosch (2000)	✓				
Pohl et al. (2005)	✓			✓	
Chen et al. (2005)		✓		✓	✓
Guerra et al. (1996)				✓	
Kang et al. (1998)	✓				
Czarnecki & Eisenecker (2000)	✓		✓	✓	✓
Batory et al. (2004)		✓			
Czarnecki et al., 2005			✓	✓	✓
Apel et al. (2013)	✓			✓	
Rodríguez et al. (2018)			✓	✓	✓
Classen, et al. (2008)	✓				
Zave (2003)	✓				
Batory (2004)	✓				
Batory et al. (2006)		✓			
Kästner et al. (2008)			✓	✓	✓
Dit et al. (2013)			✓	✓	
Apel et al. (2007)	✓		✓	✓	✓
Apel & Kästner (2009)	✓				

Fig. 2. Categorization of the 19 definitions, based on abstraction level and viewpoint.

misinterpreting technical characteristics, but would be an inaccurate description. The definitions do not necessarily refer to a certain level of detail and abstract definitions can still provide a detailed description of the term *feature*.

The role of RE versus SA appears crucial in this concept definition study. As Shekaran et al. explain the role of SA in RE by referring to RE as being concerned with the 'shape of the problem space', while SA focuses on the 'shape of the solution space' [29]. The distinction between problem and solution space is already present in the categorization, given the fact that the description of the problem space is considered an abstract characteristic and, on the other hand, the solution space is considered a technical characteristic [22]. To strengthen this reasoning, the Quality User Story (QUS) framework is in agreement stating that a user story (US) should be problem-oriented, meaning that "*a user story only specifies the problem, not the solution to it*" [24]. Moreover, Hofmeister et al. mention that architecture solutions help move the design from the problem space (in which architecturally significant requirements (ASRs) are formulated) to the solution space [18]. Splitting the definitions into two main categories can make selecting a definition easier, depending on the purpose for and context in which it is used. However, problem-space definitions (RE) can arguably be considered of higher quality or more useful, based on research by Berger et al. They state that "*good features need to precisely describe customer-relevant functionality*" [8]. Moreover, this would mean that definitions which include the customer viewpoints are more suitable in RE than those that do not.

4 Concept Definition Guidelines and Conclusion

Addressing our main research question *"How are features defined for different purposes in the context of information science literature?"*, we could not find a definitive answer. Many definitions of the term exist and one is not necessarily better or more accurate than the next. The selection of a definition is clearly dependent on the chosen perspective, and there is a wide difference in the adoption of a particular definition as derived from the citation count.

This paper distinguishes the definitions based on the research topics feature-oriented software, feature-oriented specifications, generative programming, software product lines, feature modeling, requirements engineering and release planning. If one definition had to be selected, it would have to be that of Kang et al. (1990) [20]. This is the oldest, it has been cited most frequently, and it is referenced more often by like-minded researchers than the other works included in this study.

However, the meaning of the term *feature* is largely dependent on its purpose, be it for requirements, architecture, development, modeling, target audience or otherwise. To complicate matters further, the viewpoint can influence the definition. Besides that, in this research a distinction was made between problem-oriented (abstract) and solution-oriented (technical). The only aid that can be provided when selecting a definition is the popularity of the definition, the research field and/or context, the intended viewpoint and audience. Even then, multiple options may be available.

With all this taken into account, the following guidelines are most fitting. First consider the research topic and select a definition that fits the topic to be written about. If that topic has multiple definitions, choose the definition with the most citations relative to its publication year.

For the topic feature-oriented software, the recommendation would be *"a prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems"* from Kang et al. (1990). For feature-oriented specification, it is *"an optional or incremental unit of functionality"* from Zave (2003).

For generative programming, it is *"a distinguishable characteristic of a concept (e.g., system, component, and so on) that is relevant to some stakeholder of the concept"* from Czarnecki & Eisenecker (2000). For the research topic of software product lines, the definition *"an end-user visible characteristic of a system"* from Pohl et al. (2005) should be used. For feature modelling, *"a system property that is relevant to some stakeholder and is used to capture commonalities or discriminate among systems in a family"* from Czarnecki et al. (2005). For the topic requirements engineering, *"a triplet, f = (R,W,S), where R represents the requirements the feature satisfies, W the assumptions the feature takes about its environment and S its specification"* from Classen et al. (2008). Lastly, for the research topic of release planning, *"represent needs that are gathered via meetings with customers or other stakeholders, which, once selected, are refined during a requirements elicitation process"* from Rodríguez et al. (2018) should be used.

If the specific topic is not present in Table 4, we recommend to look at the corresponding viewpoint and to choose the most fitting definition for that viewpoint based on Fig. 2. Opt for the higher level of abstraction when talking about RE, and for the technical abstraction level when talking about SA. When there are no definitive viewpoints used, work with the definition that is most all-encompassing and relatively includes most of the most frequently used terms. So, the definition we advise would be "*a unit of functionality of a software system that satisfies a requirement, represents a design decision, and provides a potential configuration option*" by Apel and Kästner [4].

Future work could look at the use of the concept *feature* in practice rather than in the literature. Perhaps, investigating whether or not features are used differently in open source and industrial software projects or in relatively large or small software development departments could yield interesting insights.

Applications of the Concept Definition Review process to other concepts in the domain of information science, computer science, or artificial intelligence would possibly reveal the plethora of concept definitions. Agreement and standardization will assist researchers to read and understand concepts better in order to utilize them in presenting and explaining their scientific contributions.

Acknowledgements We would like to thank Sven Apel for clarifying and discussing the categorization of definitions on the level of abstraction.

References

1. Andam, B., Burger, A., Berger, T., Chaudron, M.: Florida: Feature location dashboard for extracting and visualizing feature traces. In: Proceedings of the Eleventh International Workshop on Variability Modelling of Software-intensive Systems pp. 100–107 (2017)
2. Apel, S., Batory, D., Kästner, C., Saake, G.: Feature-Oriented Software Product Lines: Concepts and Implementation. Springer, Berlin, Heidelberg (2013)
3. Apel, S., Lengauer, C., Batory, D., Möller, B., Kästner, C.: An algebra for feature-oriented software development. Department of Informatics and Mathematics, University of Passau, Tech. Rep. MIP-0706 (2007)

4. Apel, S., Kästner, C.: An overview of feature-oriented software development. Journal of Object Technology **8**(5), 49–84 (2009)
5. Batory, D., Sarvela, J., Rauschmayer, A.: Scaling step-wise refinement. IEEE Transactions on Software Engineering **30**(6), 355–371 (2004)
6. Batory, D.: Feature-oriented programming and the ahead tool suite. In: Proceedings of the 26th International Conference on Software Engineering, pp. 702–703 (2004)
7. Batory, D., Benavides, D., Ruiz-Cortes, A.: Automated analysis of feature models: challenges ahead. Communications of the ACM **49**(12), 45–47 (2006)
8. Berger, T., Lettner, D., Rubin, J., Grünbacher, P., Silva, A., Becker, M., Chechik, M., Czarnecki, K.: What is a feature? A qualitative study of features in industrial software product lines. In: Proceedings of the 19th International Conference on Software Product Lines. pp. 16–25 (2015)
9. Bosch, J.: Design and use of software architectures: Adopting and evolving a product-line approach. Pearson Education (2000)
10. Chen, K., Zhang, W., Zhao, H., Mei, H.: An approach to constructing feature models based on requirements clustering. In: Proceedings of the 13th IEEE International Conference on Requirements Engineering, pp. 31–40 (2005)
11. Classen, A., Heymans, P., Schobbens, P.: What's in a feature: A requirements engineering perspective. In: Proceedings of the 11th International Conference on Fundamental Approaches to Software Engineering, pp. 16–30 (2008)
12. Creswell, J.W., Creswell, J.D.: Research design: Qualitative, quantitative, and mixed methods approaches. Sage publications (2017)
13. Czarnecki, K., Eisenecker, U.: Generative programming: methods, tools, and applications (Vol. 16). Reading: Addison Wesley (2000)
14. Czarnecki, K., Helsen, S., Eisenecker, U.: Formalizing cardinality-based feature models and their specialization. Software Process: Improvement and Practice **10**(1), 7–29 (2005)
15. Dit, B., Revelle, M., Gethers, M., Poshyvanyk, D.: Feature location in source code: A taxonomy and survey. Journal of Software: Evolution and Process **25**(1), 53–95 (2013)
16. Geeraerts, D.: Meaning and definition. In: van Sterkenburg, P. (ed.) A Practical Guide to Lexicography. John Benjamins Publishing Company (2003)
17. Guerra, S., Ryan, M., Sernadas, A.: Feature-oriented specifications. In: Proceedings of the ModelAge Workshop (1996)
18. Hofmeister, C., Kruchten, P., Nord, R., Obbink, H., Ran, A., America, P.: A general model of software architecture design derived from five industrial approaches. Journal of Systems and Software **80**(1), 106–126 (2007)
19. Kang, K., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M.: FORM: A feature-oriented reuse method with domain-specific reference architectures. Annals of Software Engineering **5**(1), 143–168 (1998)
20. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (FODA) feasibility study. Tech. rep., Carnegie-Mellon University, Software Engineering Institute (1990)
21. Kästner, C., Apel, S., Kuhlemann, M.: Granularity in software product lines. In: Proceedings of the 30th International Conference on Software Engineering, pp. 311–320 (2008)
22. Kästner, C., Thum, T., Saake, G., Feigenspan, J., Leich, T., Wielgorz, F., Apel, S.: FeatureIDE: A tool framework for feature-oriented software development. In: Proceedings of the 31st International Conference on Software Engineering. pp. 611–614. IEEE (2009)

23. Krüger, J., Gu, W., Shen, H., Mukelabai, M., Hebig, R., Berger, T.: Towards a better understanding of software features and their characteristics: a case study of marlin. In: Proceedings of the 12th International Workshop on Variability Modelling of Software-Intensive Systems pp. 105–112 (2018)
24. Lucassen, G., Dalpiaz, F., van der Werf, J., Brinkkemper, S.: Improving agile requirements: the quality user story framework and tool. Requirements Engineering **21**(3), 383–403 (2016)
25. Okoli, C., Schabram, K.: A guide to conducting a systematic literature review of information systems research. Sprouts: Working Papers on Information Systems **10**(26) (2010)
26. Pohl, K., Böckle, G., van der Linden, F.: Software product line engineering: Foundations, principles and techniques. Springer Science & Business Media (2005)
27. Rodríguez, P., Mendes, E., Turhan, B.: Key stakeholders' value propositions for feature selection in software-intensive products: An industrial case study. IEEE Transactions on Software Engineering **46**(12), 1340–1363 (2018)
28. Shaw, M.L., Gaines, B.R.: Comparing conceptual structures: Consensus, conflict, correspondence and contrast. Knowledge acquisition **1**(4), 341–363 (1989)
29. Shekaran, C., Garlan, D., Jackson, M., Mead, N., Potts, C., Reubenstein, H.: The role of software architecture in requirements engineering. In: Proceedings of the First International Conference on Requirements Engineering, pp. 239–245 (1994)
30. Spijkman, T., Molenaar, S., Dalpiaz, F., Brinkkemper, S.: Alignment and granularity of requirements and architecture in agile development: A functional perspective. Information and Software Technology **133**, 106535 (2021)
31. UCFMapper: The various types of definitions (2021), https://www.ucfmapper.com/education/various-types-definitions/
32. Whiteley, C.: Meaning and ostensive definition. Mind **65**(259), 332–335 (1956)
33. Zave, P.: An experiment in feature engineering. In: McIver, A., Morgan, C. (eds.) Programming methodology, pp. 353–377 (2003)
34. Zhang, J., Wang, Y., Xie, T.: Software feature refinement prioritization based on online user review mining. Information and Software Technology **108**, 30–34 (2019)