On the historical semantics of the notion of software architecture

S GRUNER[1]

Abstract. This study outlines in some detail the semantic variety of the notion of 'software architecture' in the field of software engineering since the early 1970s. This paper shows that there are two schools of thought in software architecture, namely a *material-substantialist* and a *formal-structuralist* one. In the former school of thought, software architecture is basically regarded as a thing (device) on its own, whereas in the latter one, software architecture is basically considered a property, not a thing (device). From an ontological point of view, these two opinions are mutually exclusive. In their mutual exclusivity, however, they coincide with non-formalist versus formalist philosophies and interpretations of informatics or computer science in general, wherein software engineering –and, by implication, software architecture– is embedded. In this way, the field of software architecture mirrors an ongoing science-philosophical dispute about the characteristics and foundations of computer science or informatics as a scholarly and practical discipline. In summary it seems fair to say that the metaphor of 'architecture', with its distinct fine arts connotations, has been particularly attractive to software engineers because this metaphor has helped software engineers to circumvent the notorious scientific immaturity and shortage of classical engineering methods in the field of software engineering.

Keywords: software architecture, historical semantics, discourse analysis

Disciplines: Computer science, informatics, history of science and technology, philosophy of science

1. Introduction

Well-known texts by Thomas Kuhn have described how young scientific disciplines typically go through the pre-paradigmatic phase of a *proto-science*,[2] which is characterized (amongst

[1] *Stefan Gruner*, Department of Computer Science, University of Pretoria, Lynnwood Road, Republic of South Africa. E-mail: sg@cs.up.ac.za

Acknowledgments: I owe many thanks to Manfred Nagl from whom I have learned much about software architecture during my time in his group at the RWTH Aachen. A few years ago he also gave an inspiring talk on the concepts of software architecture during a lecturing tour to several universities in South Africa. More recently an academic discussion with Fritz Solms (also mentioned in the acknowledgments section of his paper [51]) about the essence of software architecture motivated me to conduct this study in historical semantics. Many thanks also to Martin Olivier for ongoing science-philosophical conversations about software engineering's scientific-ness and related problems of scientific methodology. Last but not least many thanks to the anonymous reviewers of this journal for their critical comments, as well as to Cecilia du Toit for editorial services (English language advice).

[2] The term proto-science in the work of Kuhn differs in meaning from the term used in the German school of Constructivism: According to Kuhn's writings, 'proto-science' is a historic

other features) by a peculiar terminological variability and vagueness of its defining concepts and notions. Such early volatility only begins to stabilize after a pre-paradigmatic proto-science has made a historic transition into a paradigmatic science.

Software engineering is still a rather young discipline,[3] with less than fifty years of existence since the seminal NATO Science Conference in Garmisch during which the term 'software engineering' was coined. If this is so, and if Kuhn's characterization of proto-sciences was not completely wrong, then one should be able to detect significant temporal volatility in the meaning of some of software engineering's key concepts and notions during a relatively short period of time. One of these key notions –and the topic of this survey– is the notion of 'software architecture', which was initially borrowed as a metaphor from the realm of the arts and crafts of building (i.e.: house-construction and civil engineering). The purpose of using such a metaphor was to make intelligible what is *de facto* invisible (because of to the immateriality of software as a technical 'quasi-thing') – see Weingarten [57] for comparison.

Historical semantics is a branch of the history-sciences and has overlaps with philology and philosophy (amongst others). Scholars in this field study how the meanings of words and notions change during the course of time: sometimes gradually with only subtle nuances, sometimes suddenly with stark contrasts. The methods of historical semantics can be applied to any term, including the notion of 'science', and even reflexively (with self-application) onto the notions of 'history' or 'semantics' themselves.

The purpose of this study is to shed some light on the historical semantics of the notion of 'software architecture' which is one of the key terms –and also a research sub-field– of the young discipline of software engineering. In this context this study aims at

- a deeper and better reflected understanding of the notion of 'software architecture' itself (particularly for the software practitioners), as well as at
- a deeper insight into the still pre-paradigmatic character of software engineering as a proto-science (particularly for the philosophers and historians of science).

One problem of most proto-scientific software engineering concepts to date is their non-quantifiability in measurable units. This shortage has considerable implications on the semantic scope of such notions. In physics and chemistry, for example, the notion of 'energy' (which has also had its own semantic history since the times of Aristotle) might well be explicated in slightly different ways in various contemporary science-books. Nevertheless, the quantifiability of energy in the unit Joule ensures that different explications of the notion of 'energy' in different books can be related to an inter-subjective reference-semantics which is based on the objective methods and procedures which must be carried out in order to obtain the Joule scalar of some observable energetic phenomenon. With the notion of 'software

notion which refers to a science in its infancy, before maturity. In the German school of Constructivism, on the other hand, 'proto-science' is a systematic notion which refers to a some science's role as methodical foundation of another science, for example the proto-science of measurement for the science of physics. In the context of this essay, the term 'proto-science' always appears in its Kuhnian meaning. For comparison see Kuhn's "Reflections on my Critics", pp. 231- in Lakatos/Musgrave (eds.): Criticism and the Growth of Knowledge, Cambridge University Press, 1970.

[3] Much literature can be found in which software engineering is characterised as neither science nor engineering at all. It is, however, not in the scope of this study to belabour this point. For the purpose of this study it is sufficient to regard software engineering as a young 'proto-science' in the above-mentioned sense.

architecture' in software engineering, however, this has hitherto been impossible. Currently it would not make any sense to say: "here is a software architecture of 7.26 Shaw" (or whatever name might be given for any such quantifiable unit). For this reason the semantic constraints, which restrict and delimit our hermeneutic degree of freedom in the interpretations of words and terms, are considerably weaker in the proto-science of software engineering than they are in established sciences such as physics or chemistry.

In the remainder of this study, 'software architecture' is understood solely as a *product* of software engineering activities, not as the technical profession of the software architects, and also not as the activity of production which finally leads to a resulting product. In practice, of course, no notion of 'software architecture' can exist independently of the self-image of the professional software architect (including his reflective perception of his daily doing) in the environment of a software producing corporation. From a science-philosophical point of view, however, a study of the correlations between the theoretical notions held by some agent and his practical activities would fall into the area of social epistemology, not into the area of historical semantics.

An anonymous reviewer of the pre-published draft of this article had suggested that the historic-semantic problem treated in this study could possibly have its roots in software architecture being a so-called *"multiple object"*, existing simultaneously in a number of different *"versions"*. To avoid such or similar misunderstandings it must be emphasized that the notion of software architecture is a *concept* rather than an object. Moreover it seems onto-logically impossible for one and the same object to exist in *N* different versions, because otherwise there would be *N* distinguishable objects in their very own existence, not only one.[4]

As far as inter- or trans-disciplinarity –the theme of this journal– is concerned, this study presents an application of methods which are typical of the history-sciences (humanities) in the field of a technical science, for the sake of concept-clarification in such technical science. According to a classification table published by Kroeze [31], this study

- attempts to promote the understanding of a complex phenomenon which cannot be contained in a single discipline;
- is 'theoretical' in its character,[5] and
- attempts to connect different methods to find an appropriate approach to the given problem.

This study's method is thus consistent with Kroeze's attempt to utilize the humanities for enriching the field of information science –and thus to

transcend the traditional boundaries of the IS discipline [31]–

under the condition that software architecture can be classified as a sub-field of the field of the information sciences (IS). In summary, this study should be interesting for both software

[4] From a metaphysical point of view, the notion of 'multiple object' touches upon the classical problem of the relation between one-ness and many-ness. Speculatively –if some metaphysical speculation is permitted in this footnote– alone God (if any) could be a Being able to transcend the fundamental ontological difference between one-ness and many-ness by which our immanence is characterized: in this world of ours, 'one' and 'many' cannot be identified with each.

[5] 'Theoretical' in the Aristotelian sense of 'θεωρια', not in the sense of a formalized mathematical-logical system.

engineers (faculty of technology) and historians of technology (faculty of humanities) in a cross-disciplinary manner.

2. Related work

The useage of the term 'architecture' as a metaphor outside the field of material buildings is not new. For example, already in 1871 one can find 'architecture' applied to the logical and lexical structure of Hegel's system of philosophy [19]. In software engineering, because the topic of software architecture is still so new (not before 1970), science-historical work on the notion of 'architecture' has only been published sporadically so far. The papers or book-chapters recapitulated below should be considered as particularly relevant. Together, in their diversity, they can contribute to a better understanding of the reasons of *why* the meaning of the term 'software architecture' was historically not univoque.

Already in 1998 a historical-semantically motivated paper on the notion of software architecture appeared in print [8]. It asked the question *why* it is so hard to define the notion of software architecture:

> *In recent years, software engineering researchers have elevated the study of software architecture to the level of a major area of study.*

See Figure 1 below for comparison.

> *A review of the published literature, however, shows quite clearly that a unified view of software architecture has not been forthcoming* [8].

Attempting to find reasons and explanations for this correctly observed phenomenon, the authors

> *argue that our problems in obtaining an acceptable definition of software architecture are due to the assumption that software systems have a (...) unique design abstraction, determinable at the early stages of the design* [8].

Such a uniqueness presumption (one software system, one software architecture) was said to have its basis on an allegedly questionable analogy drawn by software engineers between software engineering and classical disciplines of engineering (e.g.: house construction) whereby, for example, one house has only one architecture.

> *To determine the validity of this analogy, we contrast the nature and use of architecture in the traditional building process with software development to identify the differences rather than the similarities that exist* [8].

In contrast to Kruchten's unity model [32] –one software system has one software architecture, though comprising different *aspects* (logical, dynamic, physical, developmental)– the authors of [8] have (postmodern-pluralistically) elevated Kruchten's *aspects* to different software architecture*s* (of the same system) in their own right, such that one and the same software system would now possess several different architecture*s*:

> *Our conclusion is that due to the fundamental nature of the systems we construct, attempts to depict the large-scale structure of the system in an analogous manner (to) traditional building disciplines results in many different architectures. These are fundamentally different representations and not merely different views of a single whole. Moreover, each of these is equally qualified to be labelled as the system architecture with respect to the general notion of what architecture is* [8].

With their approach, however, the authors were not able to convincingly answer the question why such allegedly different architecture*s* (instead of different architectural *views*) are still architectures *of* the same software system, instead of being architectures of several entirely different software systems. Also a follow-up paper [9] by the same authors some years later could not convincingly explain the genealogy of the historically different notions of software architecture in the field.

In 1999 the authoritative *Informatics Handbook* [42] distinguished *two* meanings of the term 'software architecture', namely a broader and a narrower one. In my own translation into English (from the German original) the handbook states:

> *In a narrower sense one understands as architecture the division of a software system into its components (mostly modules), their interfaces, the processes and dependencies amongst them, as well as their required resources (...). More generally, the notion of architecture also comprises the structural, formal, not application-oriented principles and organizational forms of software. Essential properties of a software architecture are, for example, compliance with the principle of data encapsulation, module- and class constitution, as well as the extendibility of structures* [42].

Alas the handbook did not attempt to explain the historic reasons for the semantic varieties in the such-defined technical notion(s).

In the year 2001, Mary Shaw [49] presented *in nuce* a philosophy and methodology of science and research for the field of software architecture. Motivated by concerns similar to Snelting's [50], Shaw demanded in [49] a move towards higher levels of scientificness in software architecture research. From a historical perspective, her paper's assertions about the growth of the field are consistent with the picture presented in Figure 1 (below):

> *A rough estimate is provided by citation counts for papers with 'software architecture' in the title. For a sample of 2000 citations in the Research Index database (...), virtually all of the cited papers were published in 1990 or later, and there were sharp increases in the numbers of citations for papers published after 1991 and again for papers published after 1994* [49].

Her findings are consistent with the findings of this study (see below). As far as the notion of software architecture is concerned, Shaw's paper stated:

> *Software architecture work overlaps and interacts with work on software families, component-based reuse, software design, specific classes of components (...), product line and program analysis. It is not productive to attempt rigid separation among these areas* [49].

Note that Shaw did *not* claim that it would be impossible to distinguish those areas theoretically – she merely stated that it would not be productive to keep them separate in practice. Moreover she recognized a conceptual difference between software architecture in *general*, as opposed to software architecture in more specialized contexts, where she wrote that there was

> *substantial growth since 1995 and a balance between exploration of specific problems and generalization and model development* [49].

In the same year, 2001, also the above-mentioned paper [9] appeared as a science-philosophical and science-historical meta-investigation (such as Shaw's). It must thus be regarded as related work for this study on the historical semantics of software architecture, too. The authors of that paper stated:

> *The definition and understanding of software architecture and architecture views still shows considerable disagreement in the software engineering community. This paper argues that the problems we face exist because our understanding is based on specious analogies with traditionally engineered artefacts. A review of the history of ideas shows the evolution of this understanding* [9].

Moreover:

> *One reason for those differences is the lack of a universally agreed definition or even understanding of what software architecture is or should be* [9],

though I would rather consider the lack of such a definition as a symptom of (rather than a reason for) those differences. The purpose of [9] was to search for an explanation of the observed differences, and to call normatively for serious attempts to overcome such differences:

> *The problem is not that there are no answers to these problems; rather, the difficulty arises from the fact that there have been so many different answers given* [9];

see Kuhn's notion of a pre-paradigmatic proto-science for comparison. In search of explanations of the observed historic-semantic variance in the notion of software architecture, the authors suggested:

> *One thing that is obvious from the review of the literature is that the community's understanding of software architecture has evolved based on analogies with the large-scale structure of traditionally engineered systems –*

such as, for example, between software construction and house construction– though

> *we believe this understanding is in fact the source of many problems in software architecture research* [9].

In other words, the authors attacked specifically the role model of hardware engineering, to which software engineers have always aspired in order to get (eventually) recognized and acknowledged as proper engineers, too:

> *The historical development of the research community's understanding of these terms highlights their derivation from analogies with more traditional engineering disciplines* [9],

though –now with a normative slant–

> *their failure to adequately consider the differences between software development and those other disciplines requires them to now be replaced* [9].

Alas: "Earlier versions of this material" –i.e.: their own paper –

> *have elicited comments suggesting we are merely poking holes in the current understanding of software architecture without providing a legitimate alternative, and that is certainly one valid assessment* [9].

Thus, similar to the aims of this study, the purpose of [9] was not to offer their own definition but to shed some light on the historical semantics of 'software architecture' in its variety. There is, however, a considerable problem with their critique, specifically because of the non-materiality of software as an invisible scientific object. The non-materiality and invisibility of software (as a quasi-thing) forces us to speak metaphorically about it to a large extent – see [57] for comparison. If we were to completely abandon those analogies and metaphors when speaking about (immaterial) software, then –ultimately– we would be left speechless: Note that even the term 'computer', which is nowadays used so self-understandingly in the realm of information technology, was once introduced into this realm as a metaphor. Software engineers' hitherto unfulfilled desire to become publicly recognized as proper engineers has always been too strong to make them abandon their aspirations towards classical hardware engineering with its hardware-specific terminology. In a similar way, many sciences have (with good reasons) looked up to mathematics and physics as their role models of proper science, particularly because of their striving for intra-scientific theoretical coherence and unity. By contrast, the attitude of difference-celebrating postmodernism in [9] appears in a peculiar conflict with its authors' own normative call for disciplinary unity as far as a future notion of 'software architecture' is concerned.

In the year 2003, a book-chapter of more than 50 pages was published on the history of the notion of 'architecture' in various sub-fields of computer hardware and software [28]. The chapter described *ab initio* how the metaphor of 'architecture' first came into usage in the field of computer hardware, from where it exerted its influence onto the field of software engineering, with eventual repercussions back into the domain of hardware engineering. In parallel to that forth-and-back movement of the word 'architecture' between the discourses of computer hardware and computer software, the author of [28] also diagnosed a methodological forth-and-back movement in those fields: first towards a much-desired *scientification* of the architecture-related design processes, after which a counter-movement towards a new *de-scientification* of those design processes could be observed. With its subtle tendency of working against *"scienticism"* in this field, the book-chapter [28] may arguably be counted into the ranks of postmodernism in which the liberty of the *arts* is often emphasized (over and above the rigour of science-based engineering). Claiming that a strictly scientific manner of architectural modelling, in accordance with the paradigm of mathematics and the natural sciences, would be *"impossible"* in this field, the author of [28] has classified this field of enquiry as *Design Science*, which eventually leads him to the following conclusion – (my English translation from the German original):

> *This could be the deeper reason of why the architecture metaphor has remained in use, and has now also in the software community superseded the hitherto dominant role-model of engineering* [28].

In 2004, a paper on the history and historiography of software engineering [36] identified *three different role models* for the newly emerging discipline of software engineering, namely: *applied science*, *mechanical engineering*, and *industrial engineering*. The author of [36] also analysed how those envisaged (aspired) paradigms have manifested themselves in the language (metaphors) used by those software engineers who aspired towards the one or the other paradigm of applied science or engineering. Concerning the notion of architecture in such a context, the paper stated that the notion of software architecture would have served the purpose of a compromise in the form of a common *"meeting ground"* for the disagreeing sub-communities of scientists-theoreticians and engineers- practitioners within the software engineering discipline [36].

In 2012, a paper titled "What is Software Architecture" [51] also presented an analysis of various notions of software architecture:

> *Currently there is no consensus on what exactly software architecture is or where the boundary between software architecture design and application design lies. In this paper the concept of a software architecture is analyzed from a number of different perspectives* [51].

In his own literature analysis the author of [51] identified

> *three classes of software architecture definitions which view a software architecture as (1) a high level abstraction of a software system, (2) the structure of a software system or (3) the fundamental concepts, properties and principles of design of a software system* [51].

The main purpose of that paper, however, was to promote its author's own notion of 'software architecture', not to search for the reasons of the variations in the historical semantics of the notion itself.

3. Database and method

Because it is not feasible to scrutinize every software architecture paper ever published, the well-known ACM Digital Library [1] (abbr.: ACM-DL) was chosen as database, on the reasonable assumption that the ACM-DL's large size and wide availability will imply a sufficiently high degree of representativeness for the entire field of software engineering.[6] ACM-DL's built-in search function was then used in the following manner:

- To track the time-line, which is necessary for a historical study, the search key "software architecture" was entered explicitly in an AND-combination with a specific year number, e.g.: 1986, 1994, etc.
- To filter the obtained results w.r.t. their relevance and significance for the software engineering community, the most often cited paper of each year was chosen for analysis. This is a safe choice w.r.t. the purposes of this survey, because any further existing opinions on software architecture, which were hardly ever cited (or completely ignored), cannot diminish the semantic variety of the term 'software architecture' which this survey aims to show.

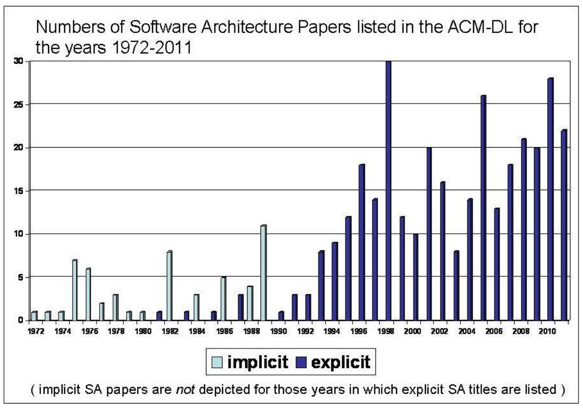
Prior to "1972" the search-term "software architecture" applied to the ACM-DL yielded no results, which is consistent with what is generally known about the history of software engineering. From "1972" upwards the search results were all non-empty. Not for all years, however, did the search engine turn out papers containing the phrase "Software Architecture" explicitly in their titles – see for comparison Shaw's paper [49] with similar considerations. For these reasons, the following cases have been distinguished:

- *Implicit* papers with "Software Architecture" only in their text bodies but not in their titles,
- *Explicit* papers with "Software Architecture" in their titles.

This distinction leads quite intuitively to the following rule (method):

- For years without any explicit "Software Architecture" papers, analyse the most often cited implicit papers.
- For years with both implicit and explicit papers, analyse the most often cited explicit papers.

[6] In future work, a study like this one could be replicated with other computer science literature databases, such as the ones provided by IEEE, AIS, CiteSeer, and the like.



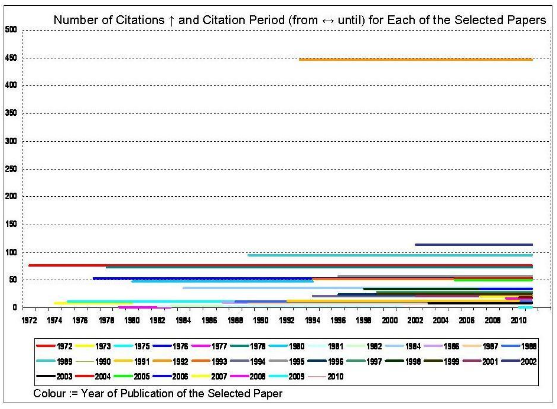


Figure 1. Database from http:/dl.acm.org/

Figure 2. Citation trails with citation counts

Figure 1 depicts clearly that:

- 1972 is the year of the first appearance of an implicit software architecture paper in the ACM-DL.
- 1981 is the year of the first appearance of an explicit software architecture paper in the ACM-DL.
- 1990 is the first year after which one can *always* find explicit software architecture papers in the ACM-DL: the research field of software architecture has been firmly established (as a sub-field of software engineering) ever since then.
- For the years 1981 to 1989, explicit software architecture titles in the ACM-DL are rather scarce and scattered: in those years the research field of software architecture had not yet been firmly established.

On the vertical ↑-axis in Figure 1 the number of papers found per year is shown –the implicit ones in pale blue, the explicit ones in dark blue– whereby existing implicit papers are *not* depicted for those years for which explicit papers could be retrieved.

Figure 2 depicts the citation *counts*, together with their citation *trails*, for each one of the most-cited publications selected for each year (according to the rule given above):

- The higher an entry on the ↑-axis in Figure 2, the more often the corresponding paper has been cited since its publication.
- The longer the line of an entry on the →-axis in Figure 2, the longer (in time) the corresponding paper has been regarded as worth mentioning by the software engineering community in its discourse.

Figure 2 clearly reveals a small number of highly cited (seminal) software architecture papers in the ACM-DL, which have apparently influenced the software architecture community's discourse quite strongly for many years. There are, however, also a number of lesser-cited papers, which have a remarkably long trail of citations from the past to nowadays, even though their accumulated number of citations is rather small. Though those papers have a rather low citation density, they are still making their impact onto the software architecture discourse nowadays.

4. Concept analysis

This section presents the results of 'mining' the selected software architecture papers from the ACM-DL in chronological (historical) order for their specific notions of 'software architecture'. In papers that do not lexically define the technical term 'software architecture', the semantics of the term had to be 'hermeneutically deciphered' as outlined in [12]. In the following, the first sub-section of this section presents the analysis of the most-cited papers, as explained above. Thereafter, the second sub-section of this section synthesizes the analysis results in such a manner that meaningful conclusions can be drawn.

4.1. Notions of software architecture in the most-cited papers of 40 years (1972-2011)

1972

In [7] a curriculum for a degree in information systems was recommended. The document did not refer explicitly to software architecture, but mentioned

> *programming and linking structures for some frequently used interface programs such as file and communication modules* [7],

with further reference to a recommended textbook by Beizer [11]. In that two-volumes book by Beizer, hardware- and software engineering were treated together, from which one may infer that the notion of 'software architecture' had not yet been much developed at the time. In analogy to hardware architecture, there was some connotation with modules, connections and interfaces in the broadest sense. Software design and software architecture were not yet precisely distinguished.

1973

In [52], the problem of software engineering in the context of operation systems (which are particular software systems themselves) was addressed, and an appeal was made to *"improve software modularity"* [52] for the sake of higher software reliability. In that paper, too, the (implicit) notion of software architecture was still linked closely to a corresponding notion of hardware architecture. The term 'software architecture' did not occur in the text body of [52], but in one of its references to a paper by Spooner [53]. In Spooner's paper, a

> *framework is described on which it should be possible to construct general purpose systems suitable for the 70's, concluding with the need for a useful method of modularizing* [53]. It *allows hitherto mandatory and untouchable components to be optional and alterable* [53].

Moreover one can find in [53] a conceptual distinction between *"logical"* and *"physical units"*, in addition to early attempts at framework standardisation, particularly by

> *developing the role of the central organizing element, as a caretaker of management decisions* [53].

1974

Paper [35] was positioned in the context of operating systems, too, whereby the notions of 'systems architecture' and 'software architecture' (in close proximity to some underlying hardware) were not sharply distinguished. Also this paper did not have the term 'software architecture' in its text body – only in a literature reference to a further paper [27] written by Gordon. The title of Gordon's paper reveals that his 'software architecture' was closely related to the design of operating systems, too. This correlation should not come as a surprise, because the discipline of software engineering was born particularly out of the practical difficulties experienced in the field of operating systems construction. A closer look into [27] reveals that its title-term 'software architecture' does not reappear in the text body of that paper. Hermeneutically one can infer that the notion of 'software architecture' in [27] referred particularly to the *data structures* needed to implement an operating system. The terms 'software architecture' and 'operating system' were thus essentially synonymous.

1975

Also in paper [58], hardware and software were regarded as tightly interwoven, with a particular focus on *"design philosophy"* [58]. Again the context was one of operating systems. In that paper, modules and components were mentioned many times, whereby it was

> *important to emphasize that we are not using the word module to mean compilation unit; rather, a module is a conceptual entity created purely to encapsulate the implementation of an abstract concept* [58].

The paper also described a concept of architectural layering (from higher levels down to lower ones in call/use-relationships) amongst the modules. Though the term 'software architecture' did not appear literally in [58], its tacit notion is obvious: the paper mentioned the concepts of modules, subsystems, logical abstraction (abstract modules), layering, as well as the correlations between the qualities of module structure and overall system reliability.

1976

Also in [21] the link between architecture and reliability in operating systems was prominent. The single term 'architecture' appeared more than 15 times in [21], though *not* the composite term '*software* architecture'. Thus the notion of 'architecture' in that paper was rather vague and not particularly software-oriented. Also in that paper a tight integration between software and hardware was emphasized. Moreover there was some semantic overlap between the notions of 'architecture' and 'software *design*'.

1977

Paper [17] shows a first turning-away from the domain of hardware-near operating systems to the domain of data structures and data types, though the overarching leitmotif of software reliability remained the same. In that paper, the word 'architecture' appeared only once in a literature reference to paper [58]; (see above). Nevertheless one can find in [17] a vague component-notion at a higher level of programming (abstract data type definition), which is conceptually related to the abstract notion of 'module' in [58].

1978

Paper [41] was written in a context of reliability issues in computing systems design, too. It also refers to the already mentioned paper [58] in its references. In [41] the term 'architecture' appeared only w.r.t. hardware, though the term 'software' appeared in [41] dozens of times. A concept of software *design*, however, can be found in close proximity to the concepts of components and interconnections, whereby the authors of [41]

> *define a system as a set of components together with their interrelationships where the system has been designed to provide a specific service. The components of the system can themselves be systems.*

The 'module' concept appears thus as synonymous to the notion of 'subcomponent' [41]. In an engineering tradition, the component concept can be found in [41] in the context of hardware, too.

1979

In [40] one can find the technical term 'software architecture' in two places. The following sentence is particularly relevant:

> *At the present time we can identify the DAT System as a software level in the software architecture of the computer network* [40].

In that paper, however, there was still confusion about the lexical notation for –as well as the systematic limits of– software architecture: A diagram on page 148 of [40] shows a diverse mixture of entity-types, from users at the top to files at the bottom, all of them depicted in rectangular boxes. In other words: software- and system architecture were not clearly distinguished from each other.

1980

Paper [29], too, references the above-mentioned paper [58], however the lexical term 'software architecture' cannot be found in [29]. Though the general term 'architecture' (without the addition of 'software') appeared often in [29], it clearly did so in the context of hardware, or systems, in general. Implicitly, however, a rudimentary notion of software architecture was expressed:

> *StarOS provides facilities for the creation of modules, each of which exports a set of functions to be invoked by the code in other modules* [29],

though there was still no clear distinction between software design and software architecture. Of particular interest in [29] is the distinction between 'physical' and 'logical' distribution:

> *We define a programmed system to be logically distributed if each component is autonomous to the extent that removal of one component will not cause the system to fail to accomplish its task in an acceptable fashion. (...) To gain increased reliability, a software system must be logically distributed* [29].

Thus, whilst not explicitly mentioning the term 'software architecture', it is obvious that there was knowledge about a correlation between overall system reliability and particular software-architectural properties, (which would be termed 'loose coupling' in later years).

1981

Paper [46] is the first paper in the ACM-DL (though not the first paper globally) which explicitly carries the term 'software architecture' in its title. The paper

> *describes an alternative approach to software architecture, where the classical division of responsibilities between operating systems, programming languages and compilers, and so forth is revised* [46].

In that paper, work-division between operating systems, compilers, and other software tools was already taken for granted. In that context the purpose of [46] was to question a conventional division of responsibilities between the major parts of software, resulting in a particular *"architectural philosophy"* [46]. However, that paper also did not provide an explicit definition of 'software architecture', and its authors soon reverted to the previously mentioned ambiguity between *software* architecture and *architecture* (in general). In [46], too, the notion of 'software architecture' was almost synonymous to the notion of 'operation system' or 'middleware' (onto which some user-application software could be planted).

1982

Paper [44] was written in the context of the then-emerging *interactive* computer systems (whereas batch systems had been dominant in the earlier past). The term 'software architecture' appeared thrice in [44]; one can infer that the term was intended to signify the data structures of a middleware system between the user and the kernel of an underlying operating system.

1983

The purpose of paper [33] was to

> *focus on the implementation of a software architecture and set of related procedures that allow the designer of a design automation tool to easily modify the control of the program and the interface to the tool's user.*

The term 'software architecture' appeared six times in the text body of that paper, though not in the form of a lexical definition. Most importantly, the following architectural requirement was stated:

> *Before designing a large system or program it is important to spend some time investigating what changes to that system are likely to occur. (...) A useful software architecture must be effective in handling these sorts of changes. The architecture described has been designed to be used in individual design tools as well as in a `shell' type program* [33].

One can thus infer that 'software architecture' was regarded in [33] as an inherent structural property of any software system itself. On the other hand, relevance of the previously encountered middleware aspect of 'software architecture' can also be found in [33], where a

> *standardized architecture that allows for the addition of new application programs without major perturbations to the rest of the system*

was mentioned.

1984

In paper [55] the term 'software architecture' appeared only once, namely in a literature reference to a paper by Deutsch [22]. Paper [55], too, was mainly concerned with hardware in combination with an operating system needed to make the hardware resources available. Deutsch's work reveals a concept of software architecture similar to the concept of a virtual machine, used as an interface between hardware and a compiler.

1985

In [10], a

> *Stream Machine is a software architecture designed to support the development and evolution, as well as the efficient execution, of software that performs both data acquisition and process control under real-time constraints.*

The word "is" is particularly interesting in this context: obviously the authors of [10] beheld software architecture from an ontological point of view as a substantial entity on its own – not as a structural property of a software. Also in that paper the term 'software architecture' appeared in close proximity to concepts of operating systems and their underlying hardware structures.

1986

Paper [45] was a follow-up publication of the already mentioned paper [55], wherein the same research group explicated the same notion of 'software architecture' (as outlined above).

1987

In paper [4], a

> *high-level Petri net model of the software architecture of an experimental MIMD multiprocessor system for Artificial Intelligence applications is derived (...). Hardware architectural constraints are then easily added (...)* [4].

Thus one can identify in [4], too, a software architecture concept at a rather low level of abstraction, in close proximity to the realm of operating systems and hardware architecture.

1988

In paper [59] the only occurrence of the term 'software architecture' can be found in a literature reference to a ten-years-older paper. Also in [59] the term 'architecture' was used in semantic proximity to notions of hardware, operating systems, and system architecture.

1989

Also in [16] the term 'software architecture' appeared only in one literature reference, namely to the already mentioned paper [22]. Context and notions in [16] were similar to the ones of [45].

1990

From the year 1990 onwards one can always find papers in the ACM-DL, which carry the term 'software architecture' explicitly in their titles. Indeed, from approximately 1990 onwards, software architecture started to be regarded as a research topic in its own right. In paper [30], nevertheless, one can still find the same connotation between the concepts of software architecture and hardware architecture as in several of the papers recapitulated above. Paper [30] stated, for example:

> *The charter of the LM-100 hardware modeller development team was (...) developing a software architecture that provided simulator and host-independence while improving overall functionality and performance.*

In such a context, software architecture was regarded once more as a software *image* of hardware structures – in [30] with the particular purpose of providing hardware simulation facilities in software.

1991

Shaw's paper [48] provided the following explication:

> *Software system design must deal with architectural issues as well as algorithms and representations. Architectural issues include gross decomposition of function, assignment of function to design elements, composition of design elements, scaling and performance, and selection among design alternatives* [48].

Subsequently several architectural styles were identified which were characterized topologically by the manner in which the modules of a software architecture are connected with each other. It is obvious that software architecture was regarded in [48] as a structural property (not as a thing or device), independent of any application context (such as operating systems or middleware).

1992

Paper [38], with its more than 440 citations, may be regarded as one of the most influential software architecture papers in the ACM-DL repository.

> *The purpose of this paper is to build the foundation for software architecture. We first develop an intuition for software architecture by appealing to several well-established architectural disciplines. On the basis of this intuition, we present a model of software architecture that consists of three components: elements, form and rationale. (...) Form is defined in terms of the properties of, and the relationships among, the elements (...). The rationale provides the underlying basis for the architecture in terms of the system constraints, which most often derive from the system requirements. We discuss the*

> *components of the model in the context of both architectures and architectural styles (...)* [38].

Moreover:

> *The 1990s, we believe, will be the decade of software architecture* –

see Figure 1 for comparison – whereby the authors explicitly declared to

> *use the term 'architecture', in contrast to 'design', to evoke notions of codification, of abstraction, of formal training (...), and of style* [38].

This was explicitly contrasted against other notions of 'architecture', particularly hardware architecture and operating systems. As far as the architectural style is concerned, the authors wrote:

> *If architecture is a formal arrangement of architectural elements, then architectural style is that which abstracts elements and formal aspects from various specific architectures. (...) In these cases, we concentrate on only certain aspects of a specific architecture (...) Given this definition of architecture and architectural style, there is no hard dividing line between where architectural style leaves off and architecture begins. We have a continuum in which one person's architecture may be another's architectural style. Whether it is an architecture or a style depends in some sense on the use* [38].

In spite of this particular flavour of perspectivism or relativism (a variant of which re-appeared twenty years later in [51]) it is obvious that the notion of 'software architecture' in [38] was a strongly topological notion at a high level of abstraction and application-independence. In this context it is also worth mentioning that the authors of [38] explicitly acknowledged the position taken by Mary Shaw – see above.

1993

Paper [2] was concerned with the formalization of hitherto informal concepts of software architecture:

> *The software architecture of most systems is described informally and diagrammatically. (...) The imprecision of these interpretations has a number of limitations. In this paper we consider these conventionalized interpretations as architectural styles and provide a formal framework for their uniform definition* [2].

In the detailed elaborations of [2] one can find (amongst others) the notions of components and connectors again, in an application- independent manner, at a high level of logical abstraction. From a theoretical point of view the specification in [2] reveals a topological notion of 'software architecture'. The example of [2] shows clearly how a mathematical-scientific approach to software engineering came together with an abstract, structural, and application-context-independent notion of 'software architecture'.

1994

In the context of *domain-specific software architecture* (DSSA), paper [54] provided a number of concise and explicit definitions for 'software architecture', 'DSSA', 'reference architecture', and 'application architecture'. The paper has thus made a valuable contribution towards the clarification of the hitherto so unclear terminology. As far as 'software architecture' (in general) is concerned, [54] defined software architecture in terms of components, connections, and constraints/rationale, with 'rationale' being understood according to [38] – see above. The infrastructure issue (so strongly emphasized later in [51]) appeared in [54] only in the context of DSSA:

Domain-Specific Software Architecture – a software architecture with reference requirements and domain model, infrastructure to support it, and process to instantiate/refine it [54].

1995

The influential paper [3], cited more than 50 times, was written by the same authors as paper [2] (see above) and should thus be regarded as a continuation thereof. The introductory section of [3] stated that software architecture

is an important level of description for software systems (...). At this level of abstraction key design issues include gross-level decomposition of a system into interacting subsystems, the assignment of function to computational components, global system properties (such as throughput and latency), and lifecycle issues (such as maintainability, extent of reuse, and platform independence) [3].

The concept of platform independence should be particularly noted here, since it was clearly in opposition to the competing concepts already mentioned, in which software architecture was (or is) regarded as tightly related to hardware structures and operating systems. Moreover one can find in [3] an awakening of historical awareness that enabled the authors to diagnose correctly that it was

only recently that software architecture has begun to emerge as a discipline of study in its own right [3].

See Figure 1 for comparison once again.

1996

In paper [33] one can find another attempt at formalising some previously informal aspects of software architecture. In particular, the aim of [33] was to remove ambiguities from informal pictorial descriptions of software-architectural ensembles. In addition to the usual static views of software architecture (including components and connections), one can find in [33] a dynamic view, too, according to which the interacting components appeared as agents.

1997

Paper [15] was written in the context of the then-emerging topic of computer-supported collaborative work (CSCW). As far as its concept software architecture in general is concerned, the paper's authors mentioned components and connectors as well as of architectural styles (patterns), such as pipes and filters, with different layers of abstraction. They also mentioned a dynamic-behavioural view (a.k.a. agents) to complement the traditional static-topological view. Work by Mary Shaw (see above) was cited in [15], too. Because of the domain-specificity (CSCW) of the approach of [15], the DSSA-related considerations of [54] –see above– are (by implication) relevant for paper [15] as well.

1998

Paper [20] (1998) presented another context- and application-specific approach to software architecture. Once more the application domain was the domain of computer networks and operating systems:

We have designed and implemented a high performance, modular, extended integrated services router software architecture in the NetBSD operating system kernel. This architecture allows code modules, called plugins, to be dynamically added and configured at run time [20].

This quotation confirms that the concept of software architecture in the realm of operating systems differed considerably from the concept of software architecture in the realm of general-purpose software engineering. What both notions had in common was merely the module concept, plus some understanding of the practical importance of architectural qualities for a given system's runtime performance.

1999

In [13] one can find an exercise in software reverse engineering. In that work, an explicit description of the (implicit) software architecture of a given software system had to be extracted. Such an extraction task is logically based on the assumption that every software system *has* its own software architecture as its structural property. Moreover one can find in [13] an important distinction between the notion of a *conceptual* architecture (i.e.: an idealised model) and the notion of a *concrete* architecture (referring to a factually existing module structure). On the basis of such a distinction, the two concepts could then be compared with each other.

2000

With its more than 50 citations the "Roadmap" paper [26] must be regarded as another milestone paper in its field. At the beginning of a new millennium the paper suggested several future research directions. The paper had an explicitly historical perspective, too:

> *Over the past decade software architecture has received increasing attention as an important subfield of software engineering* [26].

See, once again, Figure 1 for comparison. Moreover:

> *During that time there has been considerable progress in developing the technological and methodological base for treating architectural design as an engineering discipline* [26].

In that paper, 'software architecture' was explicitly defined as the *"gross organization of a collection of interacting components"* [26]. The importance of 'good' software architecture w.r.t. performance, reliability, portability, scalability and interoperability of software systems was also stated. From a historic perspective, however,

> *despite this progress, as engineering disciplines go, the field of software architecture remains relatively immature* [26],

which is typically indicated by confusion about the meaning of important technical terms – see again Kuhn's comments on the topic of pre-paradigmatic proto-sciences. Nevertheless:

> *While there are numerous definitions of software architecture, at the core of all of them is the notion that the architecture of a system describes its gross structure. This structure illuminates the top level design decisions, including things such as how the system is composed of interacting parts, (…), and what are the key properties of the parts* [26].

Moreover the paper stated:

> *Software architecture typically plays a key role as a bridge between requirements and implementation* [26],

which is closely related to the concept of software architecture as 'programming in-the-large' (PiL). Consequently, architecture description languages (ADL), in analogy to higher-level programming languages, were discussed in [26], too.

2001

Paper [43] presented a domain-specific software architecture example in the field of 'virtual reality'. In that application context, software architecture was understood to be a

> *framework for the different tasks involved (...) in virtual environments and augmented reality application* [43].

Such a framework

> *eases the development and maintenance of hardware setups in a more flexible manner* [43].

In the case of paper [43], software architecture was obviously understood (once more) as a kind of middleware, in close proximity to an operating system, between some hardware at the lower level and some user application programs at the higher level.

2002

The authors of [5] were concerned with CASE-tool-supported mappings from software architecture descriptions to executable program code, and the consistencies between those two classes of documents. Such an approach indicates clearly that software architecture was related by the authors of [5] to programming in-the-large (PiL). The term 'software architecture' was accordingly defined as

> *the organisation of a software system as a collection of components, connection between the components and constraints on how the components interact* [5].

The paper also explained an important difference between architecture description languages (ADL) and module interconnection languages (MIL):

> *ADLs differ from MILs in that the former make connectors explicit in order to describe data and control flow between components, while the latter focus on describing the uses relations between modules* [5].

Very importantly, the paper's authors also distinguished clearly between software architecture and infrastructure in the form of middle-ware, which the authors of various other papers (including [51]) had often conflated:

> *Component-based infrastructures such as COM, CORBA, and Java Beans provide sophisticated services such as naming, transactions and distribution for component-based applications. While these infrastructures do not include mechanisms for explicitly describing software architecture* [5], the example presented in that paper *shows how software architecture can be expressed in the context of component infrastructures* [5].

2003.

Paper [39], similar to [43] (see above), presented yet another domain-specific software architecture example in the field of 'virtual reality'. Not surprisingly, the concept of software architecture in [39], regarded as system-related middleware, was similar to the concept found in [43].

2004.

The authors of [14] worked with a concept of dynamic software architecture. Their paper commenced with the following –unfortunately circular– definition:

> *Dynamic software architectures modify their architecture* [14],

i.e.: a software architecture *is* something which *has* itself as a property. Obviously this does not make sense from a logical point of view. The intention of that definition, interpreted 'between the lines', should probably be rephrased such as to express that dynamic software systems modify their own architecture, or that dynamic software architectures modify their own structural properties. From [14] one can infer that the main architectural elements of such dynamic software architectures were (once more) regarded as components and connectors. Including a reference to paper [33] (see above), the remainder of paper [14] dealt with formal languages to describe the re-compositions and re-connections of such dynamic structures during the course of time.

2005

With reference to Kruchten's "4+1 Views" concept of software architecture, paper [47] focused on the development view. The paper was mainly concerned about the project-managerial aspects of software architecture construction. In such a context,

> *design rules are applied repeatedly as the system evolves, to identify violations and keep the code and its architecture in conformance with one another* [47].

The underlying concept of software architecture was basically the concept of PiL.

2006

Paper [25] presents itself in an application-specific context, once more in close proximity to operating systems and computer hardware – this time, however, in the modernised form of

> *a wearable personal monitoring service transparently embedded in user garments* [25].

By implication the software architecture concept in [25] was (once again) the middleware concept typically found in that area of research.

2007

Paper [6] presented another CASE tool supporting the management of software architecture development. Similar to the purpose of [47] (see above), the main concern of paper [6] was software project management, including the professional understanding of software architecture production processes, *not* software architecture as a product of such processes. Also in that paper, Kruchten's "4+1 Views" model was mentioned. Because paper [6] was written from an epistemological point of view, the paper neither stipulated nor implied any specific definition of the term 'software architecture'. Implicitly the usual concepts of components and connections can be inferred from [6] as the smallest common ground in this field of research. Architectural styles and patterns were mentioned, too, however independent of the question of whether or not software architecture should be regarded as context-dependent middleware.

2008

In paper [23] the concept of software architecture was once again a domain-specific one, regarded as middleware between an underlying hardware system and some higher-level application software:

> *In order to improve interoperability, we aim to identify the common traits of these systems and present a layered software architecture which abstracts these similarities by defining common interfaces between successive layers* [23].

In addition it was stated that

> *the layered architecture allows easy integration of existing software, as several alternative implementations for each layer can co-exist* [23].

In that paper, too, 'software architecture' was (more or less) a synonym for 'operating system' or 'service-providing middleware'.

2009

Paper [18]

> *proposes a high-performance network monitoring software architecture. The proposed architecture (…) is able to employ multi-core CPUs in an efficient and scalable manner".*

In the example of [18], however, the such-described software architecture had only very few modules which all implemented typical features of operating systems. From a historical perspective it is interesting to note that such a hardware-oriented understanding of 'software architecture' has survived from the early 1970s until nowadays.

2010

In [37] one can find yet another approach to CASE tool support for developing and analysing software architectures, and their properties, in the context of software engineering project management. The concern of [37] was thus, once more, the professional production of software architectures, rather than any notion of 'software architecture'. From a running example provided in [37] one can nevertheless infer a concept of software architecture which is component-based in the sense of CBSD or CBSE, whereby a component in such a context is a pre-compiled, deployable and executable unit (not identical with an un-compiled software module) which needs some middleware via which it can communicate with other components. In [37], however, such middleware was *not* be regarded as software architecture. On the contrary, 'software architecture' in the context of [37] must be understood in terms of the structural qualities of the composition of such components (on top of the necessary middleware).

2011.

Paper [56], the final one in the time-line of this survey, returns full-swing back to the context in which this survey had started with the year 1972, namely the context of education and software engineering curricula at universities. In [56] one can find a comprehensive experience report about the use of a computer game project as an illustrative example in a software architecture course at a Norwegian institution of higher education. The report had a strong focus on the use and re-use of components in a technical context of CBSE. In [56] one can thus find once more a concept of software architecture based on modules, architectural patterns, and design patterns. No clear distinction, however, was drawn between 'software architecture' and 'software design' – ditto no clear distinction between CBSE 'components' and the 'modules' of OOP. Together with a concept of different architectural views –'logical' as opposed to 'concrete'– software architecture was characterised in [56] as the structural property of any component-based software system.

4.2. Synthesis of the analysis results

As it will be shown in more detail below, the review of the 40 most often cited software architecture papers in the ACM-DL from 1972 to 2011 uncovered basically two co-existing notions of 'software architecture' during those decades:

- Software architecture beheld as a kind of *thing*, namely some type of *middleware* between some higher-level application software and some underlying hardware:
  - This *material-substantialist* concept of software architecture has been held mostly (typically) by members of the operating systems and computer hardware communities.
- Software architecture beheld as a topological *property* of *any* larger software system, independent of the software system's application context and application purpose:

- This *formal-structuralist* notion of software architecture has been held mostly (typically) by members of the general software engineering community outside the field of operating systems, particularly by theory- and science-oriented software engineers with a strong focus on formal languages and mathematical abstractions.

In what follows, those findings on the historical semantics of 'software architecture' shall be further consolidated. Such consolidation shall be achieved as follows: The characteristics identified in those 40 sample papers will be mapped onto a number of positive *is*-, or *has*-properties, P_1, P_2, P_3, etc., as well as onto a number of negative is-*not*-, or has-*not*-properties, N_1, N_2, N_3, etc. Based on these properties, in combination with those 40 papers as their carrier objects, *formal concept lattices* [24] can be constructed automatically by a software tool named *Concept Explorer*. The graphs of these concept lattices, which clearly depict inclusion, subsumption, or exclusion relations between semantic notions and concepts, are then considered to be *syndromes*, which clearly reveal the above-mentioned historic-semantic shifts in the notion of 'software architecture'.

For the totality of the 40 most-often cited papers (1972-2011) recapitulated in the previous sub-section, the following set of *positive* properties, notions and concepts in the *semantic field* of 'software architecture' has been hermeneutically identified:

- **P_1**: Basic notion of modules, connections, interfaces.
- **P_2**: Notion of sub-systems as intermediate entities 'between' modules and systems.
- **P_3**: Notion of structural principles such as layers, levels, information-hiding, and/or loose coupling.
- **P_4**: Notion of frameworks, standardisation, adaptation and re-use.
- **P_5**: Notion of architectural patterns and styles.
- **P_6**: Notion of domain-specific software architecture (DSSA).
- **P_7**: Notion of architectural views: static versus dynamic, etc.
- **P_8**: Notion of architectural refinement: abstract or conceptual versus concrete, schema versus instance, logical versus physical or technical, etc.
- **P_9**: Notion of versioning, architectural evolution or change of structure in time.
- **P_10**: Recognition of software architecture's relevance w.r.t. software maintenance.
- **P_11**: Recognition of software architecture's relevance w.r.t. runtime performance.
- **P_12**: Context of computer hardware.
- **P_13**: Context of operating systems, including networking, etc.
- **P_14**: Context of compiler construction.
- **P_15**: Context of component-based software engineering (CBSE), component-based software development (CBSD), or distributed service-oriented architecture (SOA).
- **P_16**: Software architecture as programming in-the-large (PiL).
- **P_17**: Notion of architectural language types, architecture description language (ADL) or module interconnection languages (MIL).
- **P_18**: Explicit distinction between ADL and MIL.
- **P_19**: Software architecture as operating system or virtual machine or simulated hardware architecture.
- **P_20**: Software architecture as middleware or infrastructure.
- **P_21**: Software architecture as an inherent formal topological or structural property of any software system.
- **P_22**: Software architecture as one of many possible software design alternatives, distinguished from software design as such.

- **P_23**: Software architecture as a composition of pre-fabricated re-usable components, on top of some component-supporting middleware.
- **P_24**: Explicit distinction between software architecture versus an underlying infrastructure or middleware.
- **P_25**: Explicit distinction between general software modules and components in the sense of CBSE or CBSD.
- **P_26**: Independence of the notion of software architecture from any specific application context (such as operating systems or whatever).

Likewise, for the same 40 papers as objects, the following set of *negative* properties, notions and concepts has been identified:

- **N_1**: 'Home-grown', intuitive, rather unspecific, vague or otherwise unreflected notion of software architecture, including confusion between architecture in general and software architecture in particular.
- **N_2**: Confusion between module and component, whether hardware or software, whether in source-code or pre-fabricated and pre-compiled.
- **N_3**: Confusion between the concepts of software architecture on the one hand, and systems- or computer architecture on the other hand.
- **N_4**: Confusion between software design and software architecture.
- **N_5**: Ambiguity or a lack of awareness of different types of components or modules, which all appear as equally-looking 'blobs' in sketches of (software-) architectural diagrams.

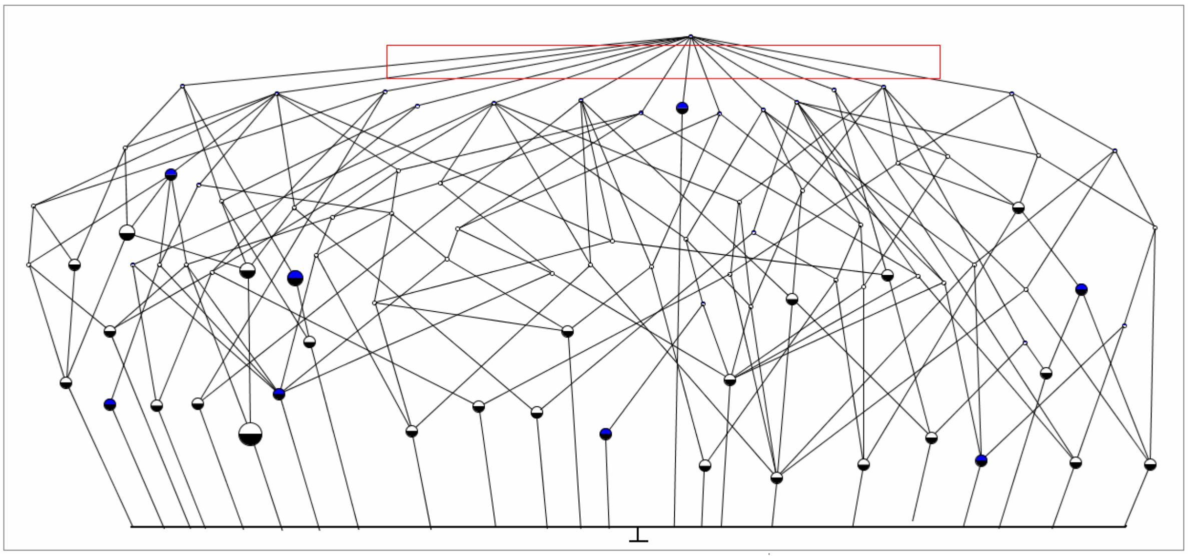


Figure 3. Lattice showing the semantic space of the most-cited software architecture papers in the ACM-DL, 1972-2011: The fourteen main conceptual branches are indicated by the red box immediately underneath the lattice's top vertex.

On the basis of all these properties, together with those 40 papers as their carrier objects, a binary object-properties-table can be constructed as described in [24], from which the lattice depicted in Figure 3 was automatically derived. The 'balls' in the figure represent the objects (i.e.: the surveyed papers), whereby a *big* ball represents a *set* of several papers that all belong to the same semantic category which is represented by such a ball. On the basis of the 40 objects of this study, the *Concept Explorer* has constructed 90 formal concepts: each of them is represented by a vertex (crossing-point) in the lattice graph of Figure 3. Not all of those 90 formal concepts are in possession of their own concrete objects. At one glance one can see that this conceptual space is wide and diverse, reflecting the multitude of differences in the historic notions of 'software architecture'. Of all these 90 concepts, the *14 main concepts*, which are represented by the 14 edges emerging immediately out of the lattice's top vertex, and which are highlighted (framed) by a red box in Figure 3, are the most important ones. Each of those is the root of a sub-lattice which has a sub-set of all 40 objects (papers) as its elements. These 14 main concepts (or sub-lattices) can be characterized as follows:

(1) is defined by P_19, with 15 objects between 1974 and 2009.
(2) is defined by P_13, with 21 objects between 1973 and 2009.
(3) is defined by N_1, with 6 objects between 1972 and 2007.
(4) is defined by P_20, with 10 objects between 1981 and 2009.
(5) is defined by N_4, with 14 objects between 1972 and 2011.
(6) is defined by P_8, with 8 objects between 1973 and 2005.
(7) is defined by P_11, with 12 objects between 1975 and 2011.
(8) is defined by P_9, with 1 object in 2004.
(9) is defined by P_2, with 5 objects between 1975 and 2011.
(10) is defined by P_7, with 6 objects between 1996 and 2011.
(11) is defined by P_21, with 10 objects between 1983 and 2011.
(12) is defined by P_4, with 3 objects between 1973 and 2000.
(13) is defined by P_5, with 10 objects between 1991 and 2011.
(14) is defined by P_26, with 9 objects between 1991 and 2007.

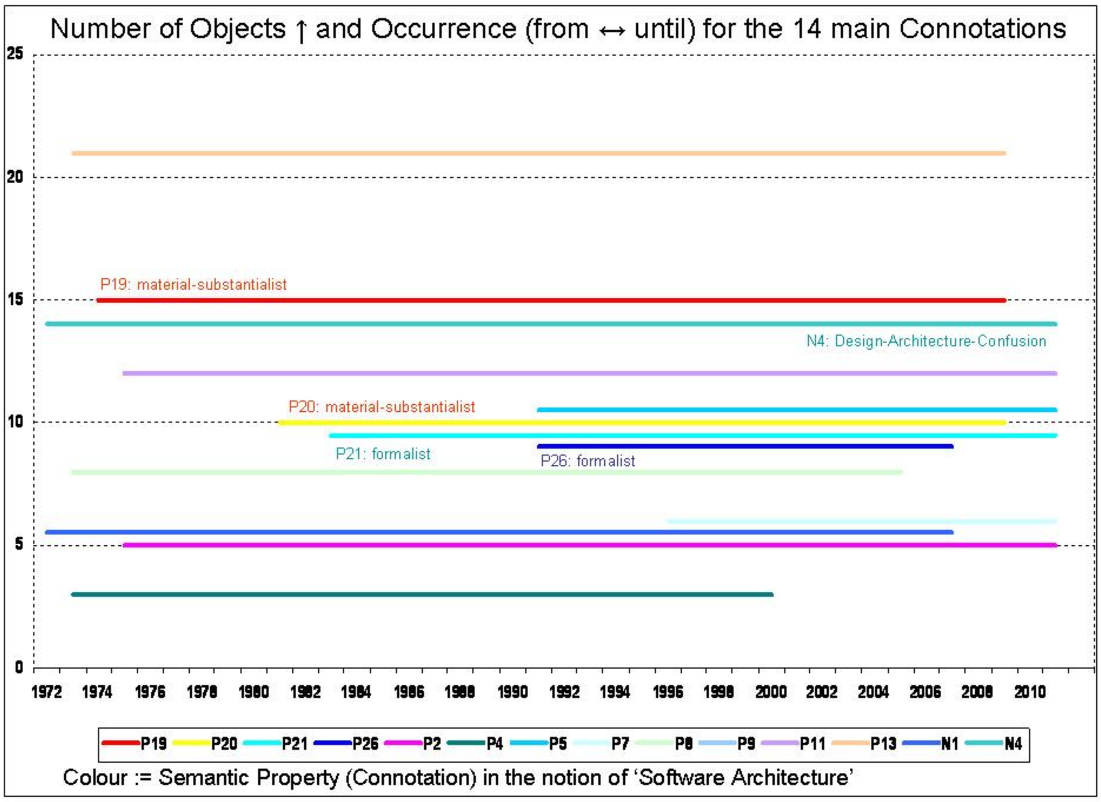


Figure 4. Historic-semantic appearance of the fourteen main connotations in the notion of software architecture, extracted from the formal concept lattice.

This list of 14 main connotations in the historical semantics of 'software architecture' is further illustrated in Figure 4, wherein the number of objects characterized by each connotation is shown on the vertical axis (↑), whilst the historical duration of that connotation is depicted as a horizontal time line (→). Also note that properties P_19 and P_20 (software architecture as operating system or virtual machine or middleware) on the one hand, and properties P_21 and P_26 (software architecture as application-independent, formal, topological property) on the other hand, are *mutually exclusive* from a logical point of view. Thereby,

- connotations P_{19,20} represent the already mentioned *material-substantialist* concept of software architecture, whereas
- connotations P_{21,26} represent the already mentioned *formal-structuralist* concept of software architecture.

These two mutually exclusive schools of thought in software architecture are also highlighted in Figure 4. Since computer science emerged from two parent disciplines –mathematics and electrical engineering– soon after the Second World War, one can still find in those two different notions of 'software architecture' the genes of the parents: whereas the formal-structuralist school of software architecture was obviously influenced by the paradigm of its mathematics parent, the material-substantialist school of software architecture was evidently influenced by its electrical engineering parent. Amongst historians of computer science and software engineering it is also well-known that those two different sources of computer science –the mathematical one and the electrical one– have never been completely confluent. The results pertaining to software architecture of this historic-semantic case study should thus be regarded as additional indicators of such a bi-polarity in the conceptual core of computer science since the earliest days of its existence.

Last but not least it is also worth noting that the negative property N_4 –confusion or ambiguity between the notions of software design and software architecture– has appeared persistently in a considerably large number of papers throughout the decades between 1972 and 2011, in spite of a more-than-sufficient availability of literature in which the difference between 'software architecture' and 'software design' had been clearly explained.

5. Conclusion and outlook

The detailed analysis unfolded in the previous sections of this study is now culminating in the following conclusions about the historical semantics of 'software architecture':

- The notion of 'software architecture' was a rather un-reflected, 'home-grown' concept in computer science and software engineering from the early 1970s until the beginning of the 1990s.
- During this early period (early 1970s to early 1990s), 'software architecture' referred in most cases to underlying computer hardware structures, operating systems, or similar kind of middleware. This semiotic relationship I have called the *material-substantialist* semantics of 'software architecture'.
- Since the beginning of the 1990s, the formerly un-reflected meaning of 'software architecture' started to become historically and philosophically reflected: the erstwhile *un*-problematic notion then became *problematic*. Philosophically and historically astute software engineers began to notice and recognize the considerable semantic varieties in the notion of 'software architecture'.
- Around the beginning of the 1990s, a new school of thought began to emerge in which 'software architecture' was defined as an abstract structural property of any large software system, independent of any specific context of application. This semiotic relationship I have called the *formal-structuralist* semantics of 'software architecture'.

- The emergence of the formal-structuralist school of thought in the field of software architecture has, however, *not* terminated the existence of the material-substantialist school of thought. Both of them have since co-existed. Further notions of 'software architecture' also arose more recently in the domain of component-based software engineering (CBSE).
- Not only did (and do) those two schools of thought simply coexist next to each other; there also arose software-methodological quarrels about which one of them should be accepted as 'right' or 'orthodox'. In Kuhnian terms one could say that the discipline of software architecture began to feel un-easy about its own pre-paradigmatic and proto-scientific state of affairs. Those methodological quarrels, as far as they continue, seem to indicate that some aspirations towards a *unified* meta-theory of software engineering are still alive, in spite of all the many postmodernist-pluralist tendencies which can be identified in nowadays meta-science. Paper [51], for example, presented from a substantialist position some of the most recent arguments against the formalist school of thought, and the authors of the somewhat earlier paper [9] had at least lamented the very existence of such conceptual discrepancies.
- From an ontological perspective, software architecture is regarded as a *thing* (in itself) by followers of the material-substantialist school of thought, whereas it is regarded as a *property* (of something else) by followers of the formal-structuralist school of thought. These two perspectives are mutually exclusive from a logical point of view.
- Amongst the analysed 40 papers published between 1972 and 2011 there was always some residual ambiguity and confusion about the related concepts of software architecture and software design. There were, however, alsothough some papers in which that confusion was explicitly addressed, and in which a clear distinction between those concepts was provided. Both 'architecture' and 'design', however, have obvious connotations with the fine *arts* and crafts, which seems to support the most recent de-scientification hypothesis postulated by Hellige for computer science as a whole [28]: followers of this third school of thought prefer to identify computer science as a so-called 'design science', in opposition to *both* the mathematical *and* the electro-technical parent paradigms at the historical roots of computer science.
- In the midst of all this conceptual confusion it has even been possible in at least one case, namely [60], for blatant pseudo-science (alluding to the metaphysics of *Feng Shui*, etc.) to make inroads into the field of software architecture, *in spite* of the well-known calls particularly by Snelting [50] and Shaw [49] for higher levels of scientific-ness in the discipline of software engineering. The peculiar Feng Shui paper [60] may thus be regarded as one particularly crass example of the de-scientification tendencies diagnosed by Hellige [28] w.r.t. the entire discipline.
- Related to the conceptual problem discussed in this study is the *non-quantifiability* of most software engineering concepts, in contrast to many scientific concepts in physics or chemistry. Whereas the understanding of those classical scientific concepts can be supported by the possibility of their objective quantification –for example: the notion of 'energy' can be objectified in units of Joule– this is to date not possible with any one of the most fundamental notions in software engineering –such as, for example: 'reliability'– to which the features of software architecture are so closely linked.
- For all these reasons I conclude that software engineering in general, including software architecture in particular, is to date still a proto-science in the Kuhnian sense of the term. From a practical perspective, however, there does not seem to be any hard incentive for the discipline of software engineering to change its proto-scientific status at all, as long as software producers are still able to sell their sub-scientifically crafted

software products profitably to accepting buyers on the commercial software market. In other words: the evolutionary transition of a pre-paradigmatic proto-science to a paradigmatic normal science does not seem to be an inevitable historic necessity.

- Last but not least I may conjecture without exaggeration that the observed bifurcation in the historical semantics of the term 'software architecture' is a heritage from the earliest days of informatics which was born out of two rather different parent disciplines, namely mathematics and electrical engineering. The dialectic tensions between *con*struction (engineering) and *ab*straction (mathematics) have always been at work in the discipline of informatics (including software engineering) which has never been theoretically unified and integrated since then. The two schools of thought in the field of software architecture, as they were identified in this study, are historically and systematically correlated with those dialectically competing and concurring pre-paradigmatic movements at the methodological foundations of that discipline.

Future work would be needed in order to further consolidate and corroborate the findings of this historic-semantic study, with literature databases other than the ACM-DL: for example the IEEE *Xplore* database, *CiteSeer*, and the like.

Epilogue – From the Analects (Conversations) of Kung Fu Tse (Confucius)

Tse-Lu spoke:

> *The lord of Wei has been waiting for you, hoping for your advice in the matters of government. What will you consider first to be done?*

Master Kung replied:

> *What is necessary is to rectify the names.*

Tse-Lu asked why.

Master Kung replied:

> *If the language is not correct, then what is said is not what is meant; if what is said is not what is meant, then what must be done remains undone; if this remains undone, morals and art will deteriorate; if justice goes astray, the people will stand about in helpless confusion. Hence there must be no arbitrariness in what is said. This matters above everything* [Analects/Conversations XIII-3].

References

[1] ACM Digital Library: *http://dl.acm.org/*

[2] Abowd, G. & Allen, R. & Garlan, D.: *Using Style to understand Descriptions of Software Architecture*. Proceedings SIGSOFT'93: 1st ACM SIGSOFT Symposium on Foundations of Software Engineering, pp. 9-20, 1993.

[3] Abowd, G. & Allen, R. & Garlan, D.: *Formalizing Style to understand Descriptions of Software Architecture*. ACM TOSEM 4/4, pp. 319-364, 1995.

[4] Ajmone-Marsan, M. & Balbo, G. & Chiola, G. & Conte, G.: *Modeling the Software Architecture of a Prototype Parallel Machine*. Proceedings SIGMETRICS'87: ACM Conference on Measurement and Modeling of Computer Systems, pp. 175-185, 1987.

[5] Aldrich, J. & Chambers, C. & Notkin, D.: *ArchJava – connecting Software Architecture to Implementation*. Proceedings ICSE'02: 24th International Conference on Software Engineering, pp. 187-197, 2002.

[6] Ali-Baba, M. & Gorton, I.: *A Tool for Managing Software Architecture Knowledge*. Proceedings SHARK-ADI'07: 2nd Workshop on Sharing and Reusing Architectural Knowledge, Architecture Rationale, and Design Intent, pp. 11-18, 2007.

[7] Ashenhurst, R.L.: *Curriculum Recommendations for Graduate Professional Programs in Information Systems*. Communications of the ACM 15/5, pp. 363-398, 1972.

[8] Baragry, J. & Reed, K.: *Why is it so hard to define Software Architecture?* Proceedings APSEC'98: Asia-Pacific Conference on Software Engineering, pp. 28-36, 1998.

[9] Baragry, J. & Reed, K.: *Why we need a different View of Software Architecture*. Proceedings IEEE/IFIP Working Conference on Software Architecture, pp. 125-134, 2001.

[10] Barth, P. & Guthery, S. & Barstow, D.: *The Stream Machine – a Data Flow Architecture for Real-Time Applications*. Proceedings ICSE'85: 8th International Conference on Software Engineering, pp. 103-110, 1985.

[11] Beizer, B.: *The Architecture and Engineering of Digital Computer Complexes*, Volumes 1 & 2. Plenum Press, 1971.

[12] Bochenski, I.M.: *Historische Methode*. In Bochenski, I.M. (1954): Die zeitgenössischen Denkmethoden. 8th ed., chapter 22, pp. 130-137, Series UTB Vol.6, Francke-Verlag, 1980.

[13] Bowman, I.T. & Holt, R.C. & Brewster, N.V.: *Linux as a Case Study – its extracted Software Architecture*. Proceedings ICSE'99: 21st International Conference on Software Engineering, pp. 555-563, 1999.

[14] Bradbury, J.S. & Cordy, J.R. & Dingel, J. & Wermelinger, M.: *A Survey of Self-Management in Dynamic Software Architecture Specifications*. Proceedings WOSS'04: 1st ACM SIGSOFT Workshop on Self-managed Systems, pp. 28-33, 2004.

[15] Calvary, G. & Coutaz, J. & Nigay, L.: *From Single-User Architectural Design to PAC* – a Generic Software Architecture Model for CSCW*. Proceedings CHI'97: SIGCHI Conference on Human Factors in Computing Systems, pp. 242-249, 1997.

[16] Chambers, C. & Ungar, D.: *Customization – Optimizing Compiler Technology for SELF, a dynamically types Object-Oriented Programming Language*. Proceedings PLDI'89: ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 146-160, 1989.

[17] Claybrook, B.G.: *A Facility for defining and manipulating Generalized Data Structures*. ACM Transactions on Database Systems 2/4, pp. 370-406, 1977.

[18] Dashtbozorgi, M. & Azgomi, M.A.: *A scalable Multi-Core-aware Software Architecture for high-performance Network Monitoring*. Proceedings SIN'09: 2nd International Conference on Security of Information and Networks, pp. 117-122, 2009.

[19] Davidson, T.: *The Logical Question in Hegel's System – Trendelenburg on Hegel's System – translated from the German of Trendelenburg*. The Journal of Speculative Philosophy 5/4, pp. 349-359, 1871. [http://www.jstor.org/stable/25665768]

[20] Decasper, D. & Dittia, Z. & Parulkar, G. & Plattner, B.: *Router Plugin – a Software Architecture for Next Generation Routers*. Proceedings SIGCOMM'98: ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, pp. 229-240, 1998.

[21] Denning, P.J.: *Fault-Tolerant Operating Systems*. ACM Computing Surveys 8/4, pp. 359-389, 1976.

[22] Deutsch, L.P.: *The Dorado Smalltalk-80 Implementation – Hardware Architecture's Impact on Software Architecture*, 1983.

[23] Echtler, F. & Klinker, G.: *Multitouch Software Architecture*. Proceeding NordiCHI'08: 5th Nordic Conference on Human- computer Interaction – Building Bridges, pp. 463-466, 2008.

[24] Ganter, B. & Wille, R.: *Formale Begriffsanalyse – Mathematische Grundlagen*. Springer-Verlag, 1996.

[25] Ganti, R.K. & Jayachandran, P. & Abdelzaher, T. & Stankovic, J.A.: *SATIRE – a Software Architecture for smart AtTIRE*. Proceedings MobiSys'06: 4th International Conference on Mobile Systems, Applications and Services, pp. 110-123, 2006.

[26] Garlan, D.: *Software architecture - a Roadmap*. Proceedings Future of Software Engineering, pp. 93-101, 2000.

[27] Gordon, R.L.: *The Impact of Automated Memory Management on Software Architecture*. Computer (IEEE) 6/11, pp. 31-36, 1973.

[28] Hellige, H.D.: *Die Genese von Wissenschaftskonzepten der Computerarchitektur – vom 'System of Organs' zum Schichtenmodell des Designraums*. In Hellige, H.D. (ed.), Geschichten der Informatik: Visionen Paradigmen Leitmotive, pp. 411-463, Springer-Verlag, 2003.

[29] Jones, A.K. & Schwarz, P.: *Experience Using Multiprocessor Systems – a Status Report*. ACM Computing Surveys 12/2, pp. 121-165, 1980.

[30] Kelly, N.F. & Stump, H.E.: *Software Architecture of Universal Hardware Modeler*. Proceedings EURO-DAC'90: Conference on European Design Automation, pp. 573-577, 1990.

[31] Kroeze, J.H.: *Transdisciplinarity in IS – The next Frontier in the Computing Disciplines*. Sprouts Working Papers on Information Systems 12/2, 2012. [http://sprouts.aisnet.org/12-2]

[32] Kruchten, P.B.: *The 4+1 View Model of Architecture*. IEEE Software 12/6, pp. 42-50, 1995.

[33] Le-Métayer, D.: *Software Architecture Styles as Graph Grammars*. Proceedings SIGSOFT'96: 4th ACM SIGSOFT Symposium on Foundations of Software Engineering, pp. 15-23, 1996.

[34] Leath, C.L. & Ollanik, S.J.: *Software Architecture for the Implementation of a Computer-Aided Engineering System*. Proceedings DAC'83: 20th Design Automation Conference, pp.137-142, 1983.

[35] Löfgren, L.: *Reference Concepts in a Tree-structured Address Space*. ACM SIGARCH Computer Architecture News 3/4, pp. 71-79, 1974.

[36] Mahoney, M.S.: *Finding a History for Software Engineering*. IEEE Annals of the History of Computing 26/1, pp. 8-19, 2004.

[37] Martens, A. & Koziolek, H. & Becker, S. & Reussner, R.: *Automatically improve Software Architecture Models for Performance, Reliability, and Cost using Evolutionary Algorithms*. Proceedings WOSP/SIPEW'10: 1st joint International Conference on Performance Engineering, pp. 105-116, 2010.

[38] Perry, D.E. & Wolf, A.L.: *Foundations for the Study of Software Architecture*. ACM SIGSOFT SEN 17/4, pp. 40-52, 1992.

[39] Piekarski, W. & Thomas, B.H.: *An Object-Oriented Software Architecture for 3D Mixed Reality Applications*. Proceedings ISMAR'03: 2nd IEEE/ACM International Symposium on Mixed and Augmented Reality, pp. 247-256, 2003.

[40] Popescu-Zeletin, R.: *The Data Access and Transfer Support in a Local Heterogeneous Network (HMINET)*. Proceedings SIGCOMM'79: 6th Symposium on Data Communications, pp. 147-152, 1979.

[41] Randell, B. & Lee, P. & Treleaven, P.C.: *Reliability Issues in Computing System Design*. ACM Computing Surveys 10/2, pp. 123-165, 1978.

[42] Rechenberg, P. & Pomberger, G. (eds.): *Informatik-Handbuch*, 2nd edition. Hanser-Verlag, 1999.

[43] Reitmayr, G. & Schmalstieg, D.: *An open Software Architecture for Virtual Reality Interaction*. Proceedings VRST'01: ACM Symposium on Virtual Reality Software and Technology, pp. 47-54, 2001.

[44] Relles, N. & Sondheimer, N.K. & Ingargiola, G.P.: *Recent Advances in User Assistence*. ACM SIGSOC Bulletin 13/2-3, pp. 1-5, 1982.

[45] Samples, A.D. & Ungar, D. & Hilfinger, P.: *SOAR – Smalltalk without Bytecodes*. Proceedings OOPSLA'86: Conference on Object-Oriented Programming Systems, Languages and Applications, pp. 107-118, 1986.

[46] Sandewall, E. & Strömberg, C. & Sörensen, H.: *Software Architecture based on Communicating Residential Environments*. Proceedings ICSE'81: 5th International Conference on Software Engineering, pp. 144-152, 1981.

[47] Sangal, N. & Jordan, E. & Sinha, V. & Jackson, D.: *Using Dependency Models to manage Complex Software Architecture.* Proceedings OOPSLA'05: 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications, pp. 167-176, 2005.

[48] Shaw, M.: *Heterogeneous Design Idioms for Software Architecture*. Proceedings IWSSD'91: 6th International Workshop on Software Specification and Design, pp. 158-165, 1991.

[49] Shaw, M.: *The Coming-of-Age of Software Architecture Research*. Proceedings ICSE'01: 23rd International Conference on Software Engineering, pp. 656-664, 2001.

[50] Snelting, G.: *Paul Feyerabend und die Softwaretechnologie*. Softwaretechnik-Trends 17/3, pp. 55-59, 1997.

[51] Solms, F.: *What is Software Architecture?* Proceedings SAICSIT'12: Annual Symposium of the South African Institute for Computer Science and Information Technology, pp. 363-373, 2012.

[52] Spier, M.J. & Hastings, T.N. & Cutler, D.N.: *An experimental Implementation of the Kernel/Domain Architecture*. Proceedings SOSP'73: 4th ACM Symposium on Operating System Principles, pp. 8-21, 1973.

[53] Spooner, C.R.: *A Software Architecture for the 70's: Part I – The general Approach*. Software Practice & Experience 1/1, pp. 5-37, 1971.

[54] Tracz, W.: *Domain-specific Software Architecture (DSSA) frequently asked Questions (FAQ)*. ACM SIGSOFT SEN 19/2, pp. 52-56, 1994.

[55] Ungar, D. & Blau, R. & Samples, D. & Patterson, D.: *Architecture of SOAR – Smalltalk on a RISC*. Proceedings ISCA'84: 11th Annual International Symposium on Computer Architecture, pp. 188-197, 1984.

[56] Wang, A.I.: *Extensive Evaluation of Using a Game Project in a Software Architecture Course*. ACM TOCE 11/1, Art. 5, pp. 5:1-5:28, 2011.

[57] Weingarten, R.: *Metaphern in der Fachkommunikation und der Modellbildung in der Informatik*. In Schefe, P. & Hastedt, H. & Dittrich, Y. & Keil, G. (eds.), Informatik und Philosophie, pp. 279-293, B·I·Wissenschaftsverlag, 1993.

[58] Wulf, W.A.: *Reliable Hardware-Software Architecture*. Proceedings International Conference on Reliable Software, pp. 122-130, 1975.

[59] Wybranietz, D. & Haban, D.: *Monitoring and Performance-Measuring Distributed Systems during Operation*. Proceedings SIGMETRICS'88: Conference on Measurement and Modeling of Computer Systems, pp. 197-206, 1988.

[60] Zarayaraz, G. & Rodrigues, P. & Thambidurai, P. & Kuppuswami, S.: *A New Approach to Software Architecture*. ACM SIGSOFT SEN 28/2, Art. 17, pp. 17:1-17:6, 2003.