The Golden Age of Software Architecture: A Comprehensive Survey

Mary Shaw and Paul Clements*

February 2006
CMU-ISRI-06-101

Institute for Software Research International
School of Computer Science
5000 Forbes Avenue
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

This retrospective on nearly two decades of software architecture research examines the maturation of the software architecture research area by tracing the evolution of research questions and results through their maturation cycle. We show how early qualitative results set the stage for later precision, formality, and automation, how results have built up over time, and how the research results have moved into practice.

*Software Engineering Institute, Carnegie Mellon University, Pittsburgh PA 15213

Mary Shaw's work is supported by the Software Industry Center, the A.J. Perlis Chair of Computer Science, and the National Science Foundation under Grant CCF-0438929. The Software Engineering Institute is sponsored by the U.S. Department of Defense. The views and conclusions contained in this document are those of the authors and do not necessarily reflect the opinions of the sponsoring organizations.

Keywords: Software architecture, technology maturation, history of software engineering

The Golden Age of Software Architecture: A Comprehensive Survey *

Mary Shaw
Institute for Software Research, International
Carnegie Mellon University
Pittsburgh PA 15213 USA
mary.shaw@cs.cmu.edu

Paul Clements
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213 USA
clements@sei.cmu.edu

Abstract

This retrospective on nearly two decades of software architecture research examines the maturation of the software architecture research area by tracing the evolution of research questions and results through their maturation cycle. We show how early qualitative results set the stage for later precision, formality, and automation, how results have built up over time, and how the research results have moved into practice.

Keywords: Software architecture, technology maturation, history of software engineering

1. Introduction

Since the late 1980's, software architecture research has emerged as the principled study of the large-scale structures of software systems. From its roots in qualitative descriptions of empirically observed useful system organizations, software architecture has matured to encompass broad explorations of notations, tools, and analysis techniques. Whereas initially the research area interpreted software practice, it now offers concrete guidance for complex software design and development. It has made the transition from basic research to an essential element of software system design and construction.

This retrospective examines the trajectory software architecture has taken in the context of a technology maturation model, matching significant accomplishments in software architecture to the stages of that model to gain perspective on where the field stands today.

This trajectory has taken software architecture to its "golden age" and that in the near future it will attain the status of all truly successful technologies: It will be considered an unexceptional and essential part of software system building, taken for granted, employed without fanfare, and assumed as a natural base for further progress.

** This paper updates an invited keynote for ICSE 23, "The Coming-of-Age of Software Architecture Research" by Mary Shaw[77]. It is also the basis for "The Golden Age of Software Architecture" published in IEEE Software, March/April 2006 [79].*

2. How Technologies Mature

Redwine and Riddle [71] reviewed several software technologies to see how they develop and propagate. They found it typically takes 15-20 years for a technology to enter widespread use. They identified six typical phases:

- *Basic research*. Investigate basic ideas and concepts, put initial structure on the problem, frame critical research questions.
- *Concept formulation*. Circulate ideas informally, develop a research community, converge on a compatible set of ideas, solve specific subproblems, refine the structure of the problem.
- *Development and extension*. Explore preliminary applications of the technology, clarify underlying ideas, generalize the approach.
- *Internal enhancement and exploration*. Extend approach to other domains, use technology for real problems, stabilize technology, develop training materials, show value in results.
- *External enhancement and exploration*. Similar to internal, but involving a broader community of people who weren't developers, show substantial evidence of value and applicability. Flesh out the details to provide a complete system solution.
- *Popularization*. Develop production-quality, supported versions of the technology, commercialize and market technology, expand user community

As technologies mature, their institutional mechanisms for disseminating results also change. These mechanisms begin with informal discussions among colleagues and progress to products in the marketplace. Along the way, preliminary results of the first two phases appear in position papers, workshops, and research conferences. As the ideas mature, results appear in conferences and then journals; larger conferences set up tracks featuring the technology, and eventually richer streams of results may justify topical conferences. Books that synthesize multiple results help to move the technology through the exploration phases. University courses, continuing education courses, and standards indicate the beginning of popularization.

3. Maturation of software architecture

Software architecture is the principled study of the large-scale structures of software systems. From its roots in qualitative descriptions of useful system organizations, software architecture has matured to encompass broad explorations of notations, tools, analysis techniques, and creation methods. Whereas initially the research area interpreted software practice, it now offers concrete guidance for complex software design and development.

Software architecture overlaps and interacts with the study of software families, domain-specific design, component-based reuse, software design, specific classes of components, and program analysis. It is not productive to attempt rigid separation among these areas; research can certainly contribute to more than one.

One way to see the growth of the field is to examine the rate at which earlier results serve as building blocks for subsequent results. A rough estimate is provided by citation counts for papers with "software architecture" in the title. Virtually all of the cited papers were published in 1990 or later. There were steady increases in the number of citations of papers published from 1991 to 1996 and a sharp increase for papers published in 1998. The two dozen most widely-cited books and papers were published between 1991 and 2000. They include five books ([12][17][72][82][94], 1995 to 2000), four papers presenting surveys or models for the field ([33][34][57] [68], 1992 to 1997), six papers dealing with architecture for particular domains ([18][19][24][27][51][53], 1991 to 1998), seven formalizations ([1][2][3][43][54][55][62], 1992 to 1996), and one paper each on an architectural description language [80] and an analysis technique [46]. The major changes in this pattern since a similar count in 2001 [77] are an increase in citations of formalizations and substantial turnover in the most-cited papers about architectures for specific domains.

This indicator is based on the published literature, so it naturally reflects the first three phases of development. Imperfect though this estimate may be, it still indicates very substantial growth over the past decade or so and a balance between exploration of specific problems and development of generalizations and formalizations. Of the two dozen papers that were most commonly cited in 2001, fourteen remain among the most commonly cited papers in 2005 – an indication that the seminal sources have been identified. The Appendix compares the lists from 2001 and 2005.

Here are some of the highlights of the field's development, mapped to the Redwine/Riddle model. The chronology is not as linear as the Redwine/Riddle model might suggest: different aspects of the field evolve at different rates; transitions between phases do not happen instantly; and publication dates lag the actual work by different amounts, as indicated in the figure Nevertheless, overall progress corresponds fairly well to their model.

3.1 Basic research phase: 1985-1994[1]

For as long as complex software systems have been developed, designers have described their structures with box-and-line diagrams and informal explanations. Good designers recognized stylistic commonalities among these structures and exploited the styles in ad hoc ways. These structures were sometimes called architectures, but knowledge about common styles –generally useful structural forms – was not systematically organized or taught.

Significantly, by the mid-1980s several foundational ideas were firmly in place, having traveled their own 15-20-year Redwine-Riddle cycles. These included information-hiding, abstract data types, and other ideas that contributed to considering software elements as black boxes. Object-oriented development was building on abstract data types and inheritance. These ideas all had their foundations on observations, for example by Dijkstra [26] and Parnas [64], that it was not enough for a computer program to produce the correct outcome. Other qualities of the software, such as dependability and maintainability, were also important and could be achieved by careful structuring.

In the late 1980s people began to explore the advantages of deliberately-designed specialized software structures for specific problems. Some of this work addressed software system structures for particular product lines or application domains such as avionics [66], oscilloscopes [25] and missile control [22][60].

Other work organized the informal knowledge about common formations of software structures, or architectural styles, that can be used in a variety of problem domains. This work cataloged existing systems to identify common architectural styles such as pipe-filter, repository, implicit invocation, and cooperating processes, both by identifying the architectures of specific classes of systems [7][63] and by finding general ways to describe such structures [4][74][75][76]. These complementary lines of research led to models for explaining the architectural styles and to two widely cited papers in 1992 and 1993 that established the structure (and settled the name) of the field [34][68].

[1] Time spans for phases are suggested by the dates of the cited work in the corresponding section, discounting foundational works from the 1960s and 1970s.

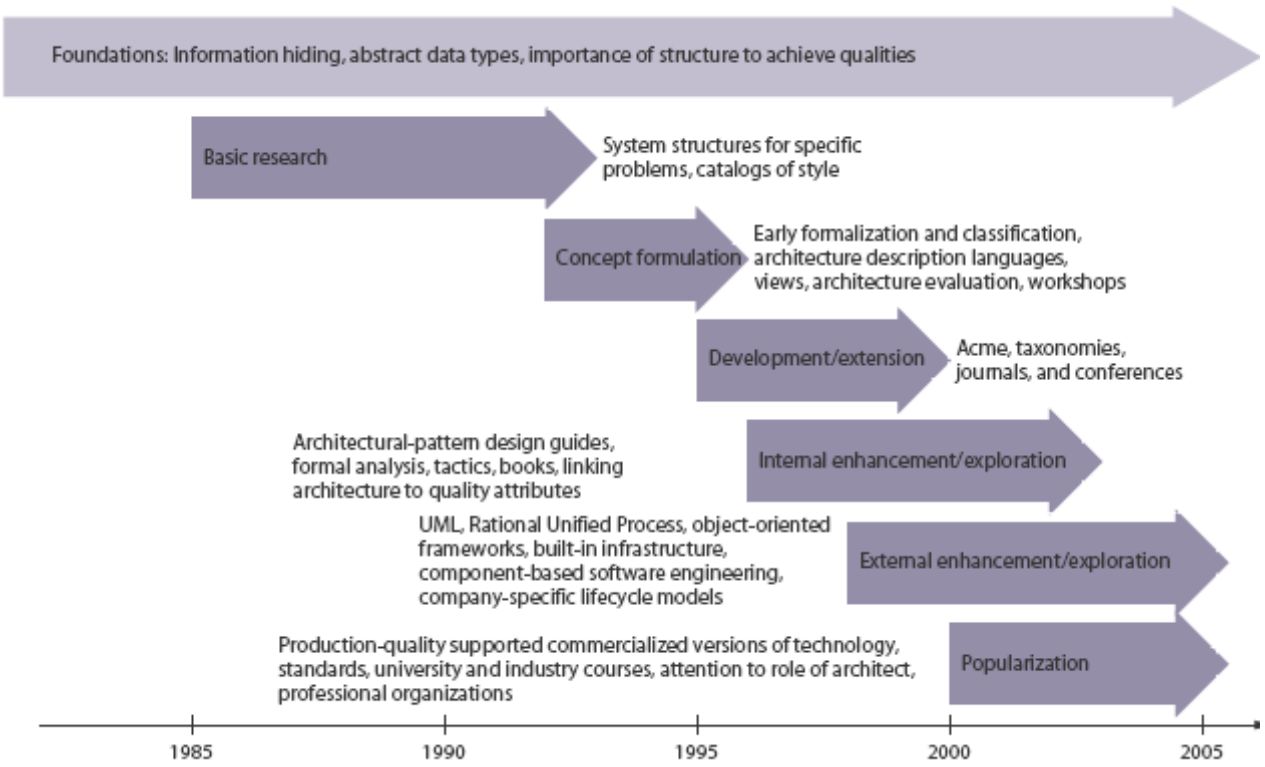

Figure 1. Maturation of the software architecture field.

3.2 Concept formulation phase: 1992-1996

The basic models were elaborated and explored largely through work in architecture description languages, early formalization, and classification. The early ideas centered on the system structures that commonly occurred in software systems, and the results emphasized description of organizations found in practice [34].

Architecture description languages [57] served as a vehicle to flesh out specific details of a variety of aspects of architecture. The early ideas centered on the system structures that commonly occurred in software systems, and the results emphasized description of organizations found in practice [34]. Ideas about system organization, especially alternatives to the then-emerging object orientation, were elaborated in programming languages. These languages included Aesop [30] (exploiting specific properties of styles), C2 [56] (exploring power of a particular event-based style), Darwin [54] (design and specification of dynamic distributed systems), Meta-H [13] (real-time avionics control), Rapide [52] (simulation and analysis of dynamic behavior), UniCon [80] (extensible set of connectors and styles, compilation to code), and Wright [4] (component interaction).

Formalizations developed in parallel with the language development. Sometimes this was integral to the language (Darwin [55], Rapide, Wright [3]), and in other cases it was more independent, as the formalization of style [1][2] or formal analysis of a specific architectural model [91][43] or application area [51][53]. The recognition that multiple views must be reconciled in architectural analysis [48] helped to frame the requirements for formalism.

The early narrative catalogs of styles were expanded in taxonomies of styles [78] and of the elements that support those styles [47]. The common forms were cataloged and explained as patterns [17][72]. An early book [82] on these ideas set the stage for further development.

Understanding of the relationship between architectural decisions and a system's quality attributes revealed software architecture validation as a useful risk-reduction strategy. Interconnectivity metrics [73], checklists for architects [8], and attribute-specific architecture analysis techniques [84] gave way to more general architecture evaluation methods such as the Software Architecture Analysis Method (SAAM) [46].

Significant in this phase was the emergence of architectural views as a working concept. Parnas set the stage for this in 1974 [65] with his observation that software systems have many structures that serve different engineering purposes and it makes little sense to call out any one as distinguished. After percolating for a Redwine-Riddle maturation period, the concept flowered in influential papers [90][48][68] that firmly established views in architectural practice.

Workshops on other topics (such as the International Workshop on Software Specification and Design) provided a temporary home for the software architecture community. A formative Dagstuhl seminar held in 1995 [32] gathered researchers to think about the layout and future directions of the field. A series of International Software Architecture Workshops (associated with other conferences) from 1995 to 2000 provided a welcome and ongoing forum devoted solely to software architecture.

3.3 Development and extension phase: 1995-2000

During this phase, the focus shifted to unifying and refining initial results. The Acme architectural interchange language began with the goal of providing a framework to move information between architecture description languages [31]; it later grew to integrate other design, analysis and development tools.

Refinement of the taxonomies of architectural elements [59] and languages [58] also continued.

The institutions of the area also matured. The IEEE's *Transactions on Software Engineering* had a special issue on software architecture in 1995[33]. The special "road-map" track at the ICSE 2000 conference included software architecture [29] among its topics to survey, and it is now routine for ICSE to have one or more sessions on architectural topics. A standalone conference, the Working IEEE/IFIP Conference on Software Architecture (WICSA) began in 1998 and continues to the present [95]. One of its sponsors is a new IFIP working group on software architecture [41].

3.4 Internal enhancement and exploration phase: 1996-2003

Architectural styles (which during this stage shifted their name to architectural patterns to acknowledge their kinship with design patterns) are commonly used informally as design guides. The explicit attention to this aspect of design is increasing, and as a result we are gaining experience.

A few formal analyses of real system designs have been done as well. For example, architectural specification of the High-Level Architecture for Distributed Simulation [5] was able to identify inconsistencies before implementation, thereby saving extensive redesign.

Architectural analysis and evaluation emerged as a fertile sub-topic. At the SEI, the Software Architecture Analysis Method [46] gave way to the Architecture Tradeoff Analysis Method (ATAM) [45], which supports analysis of the interaction among quality attributes as well as the attributes themselves. Books on the application of the research to practice [12][36] set the stage for external exploration. Books on specialized parts of the practice such as architecture evaluation [21] and documentation [20] also emerged, signaling a new kind of maturation of the overall field.

Another internal enhancement of note was the exploration of architectural tactics [9], which are fine-grained architectural design decisions that contribute to architectural patterns. During this stage, the importance of quality attributes increased, along with architecture's role in achieving them [11]. The early 2000's saw work strongly connecting quality attributes and architectural design decisions, and for the first time an automated architectural design aid seemed within reach [10].

3.5 External enhancement and exploration phase: 1998-present

Several areas have matured enough to be useful outside their developer groups.

UML [14], under the leadership of (at the time) Rational, has integrated a number of design notations and developed a method for applying them systematically. UML has, for better or (many would say) for worse, become the industry standard ADL. Tied inextricably to UML is the Rational Unified Process, a tool-centered industrialization of Kruchten's original elegant idea of 4+1 views [48]. For the most part, UML provides graphical notations; it remains, however, to provide a robust suite of tools for analysis, consistency checking, or other means of automatically connecting the information expressed in UML with the code of the system.

The rise of object-oriented software frameworks provided a rich development setting for object-style architecture and considerable public enthusiasm for object-orientedness. The benefits of the built-in infrastructure and available, interoperable components provided substantial incentive to use the frameworks even when they were not ideal fits for the problems. These satisfied needs for those architectures. As a result, work on general-purpose architecture description languages gave way to extensive support for specific architectures. At about the same time, architecture provided a solid enough foundation on which to implicitly base the component-based software engineering movement [92].

Also indicative of external enhancement are company-specific end-to-end architecture-based development lifecycle models, such as the Raytheon Enterprise Architecture Process (REAP) [69].

3.6 Popularization phase: 2000-present

The popularization phase is characterized by production-quality, supported, commercialized, and marketed versions of the technology, along with an expanded user community.

Architectural patterns, fueled in part by the explosion of the World Wide Web and web-based e-commerce, are leading the commercialization wave. N-tier client-server architectures, agent-based architectures, and Service-Oriented Architectures – along with the interfaces, specification languages, tools and development environments, and wholly implemented components, layers, or subsystems to go along with them – are examples of enormously successful architectural patterns that have entered everyone's vocabulary. Microsoft says its .NET platform "includes everything a business needs to develop and deploy a Web service-connected IT architecture: servers to host Web services, development tools to create them, applications to use them, and a worldwide network of more than 35,000 Microsoft Certified Partner organizations to provide any help you need."[61] Connected services, tools, applications, platforms, and an army of vendors, all built around an architecture: This is true popularization.

One of the hallmarks of a production-ready technology is good standards. Standards for particular component families (e.g., COM, CORBA) and interfaces (e.g., XML) have existed for several years, but they reflect component reuse interests as much as architectural interests. An ANSI/IEEE standard [39] has attempted to codify the current best practices and insights of both the systems and software engineering communities in the area of documentation. Newer standards are emerging all the time, primarily in support of the important patterns mentioned above. Recently AADL (a true architecture description language) was standardized by the Society of Automotive Engineers (SAE)[86].

One sure sign of an expanded user community is the degree to which people take ownership of the terms and concepts. Bill Gates, who could have any title he chooses, is Microsoft's "chief software architect". The OMG chose to call its development initiative separating business and application logic from platform technology "model driven architecture." The SEI invites people to submit their working definitions of "software architecture," and by late 2005, over 156 definitions had been submitted by practitioners in 24 countries [87]. Another sign is the way the term gets co-opted and diluted by people pulling their own interests under the currently popular umbrella. Terms such as "program architecture" make us shudder.

An institutional indicator of popularization is the degree to which the subject is routinely taught. In universities, software architecture is moving from graduate to undergraduate curricula; more than one textbook for introductory software engineering courses now includes a chapter on "architectural design" [70][93]. In the ACM/IEEE undergraduate software engineering curriculum [44], 20% of the software design unit is devoted to software architecture. The Software Engineering Body of Knowledge identifies software architecture as a major section in the software design chapter [38]. Industrial courses and certificate programs are also widely available (e.g., [88], [16], [37], [42]).

Finally, "software architect" is a job title that one would expect to find in any company that builds software-intensive systems, and professional organizations such as the Worldwide Institute of Software Architects [96] and the International Association of Software Architects [40] allow communication, foster networking, encourage professional practice, and (one hopes) help their members sort out the avalanche of books – over 50 – now available on the topic.

Conferences continue to thrive, not only for the research community but for user networks. In late 2005, the SEI listed 25 upcoming conferences explicitly listing "software architecture" in their calls for participation[89]. These include user network meetings as well as research conferences.

4. Current status

It is fair to say that the broad concept of software architecture has run the full course of the Redwine-Riddle model, pretty much right on schedule. The result is a breathtaking capability for reliably designing systems of unprecedented size and complexity verging on a true engineering discipline. Consider the resources readily available to a contemporary software architect:

- *Off-the-shelf industrial training and certification programs* that reflect a converging sense of what software architecture is and why it is a critical discipline
- *Standard architectures* for countless domains and applications. For example, nobody will ever again have to design from scratch a banking system, an avionics system, a satellite ground system, a web-based e-commerce system, or a host of other varieties of systems.

- Where total architectural solutions do not yet exist, partial ones certainly do in the form of *catalogs of architectural patterns and tactics* that have been used to solve a myriad of problems, many of which involve the achievement of quality attributes.
- *End-to-end lifecycle models* (industry-wide or, more likely, company-specific) that are centered on architectural principles.
- Robust and repeatable approaches to *architecture evaluation and validation*
- Practical approaches to *architecture documentation*, supported by standards for artifacts and standards for languages in which to render the artifacts.
- *Robust tool environments* to capture designs
- *Commercial-quality architectural infrastructure layers* to handle inter-component communication and coordination distributed generic computing environments
- *Commercial-quality application layers* (and tooling) to handle business logic, user interface, and support function layers
- *Career tracks* and professional societies for software architects.
- An active *pipeline of journals and conferences* devoted to software architecture, serving as a conduit between research and practice communities.

These and other indicators indicate that software architecture is integrated in the fabric of software engineering.

5. What's next?

Software engineering research is often motivated by problems that arise in the production and use of real-world software. Technical ideas often begin as qualitative descriptions of problems or practice and gradually become more precise – and more powerful – as practical and formal knowledge grow in tandem. Thus, as some aspect of software development comes to be better understood, more powerful specification mechanisms become viable, and this in turn enables more powerful technology.

We see that software architecture has followed this approach, growing from its adolescence in research laboratories to the responsibilities of maturity. This brings with it additional responsibility for researchers to show not just that new ideas are promising (a sufficient grounds to continue research) but also that they are effective (a necessary grounds to move into practice).

As a result, software architecture researchers must not be content with simply doing more research in the style of the past decade. Certainly there are new ideas yet to be explored in that form, but the last decade has opened even more opportunities in the form of research to make existing results more robust, more rigorously understood, and more ready to move into application. For example, there was a time when it seemed that a new ADL emerged almost monthly. Now someone proposing a new language has to ask themselves (or be prepared to be asked by their funding agency) "Does what you're proposing have any chance of unseating UML? What tooling will you provide with it?"

Nevertheless, there are significant opportunities for new contributions in software architecture. Some of the more promising areas seem to be:

- *Continuing to explore formal relationships between architectural design decisions and quality attributes.* This could one day lead to a practical and sophisticated automated architecture design assistant. In addition, it could enable earlier and more accurate predictions of the value a system would deliver to specific types of users.
- *Finding the right language in which to represent architectures.* UML 2.0 was a marginal improvement over its predecessor, but it still lacks basic architectural concepts such as "layer" or a faithful notion of "connector"; it lacks the ability to analyze interactions among views; it too easily mixes design concepts with implementation directives; and it lacks the ability to make strong connections to code.
- *Finding ways to assure conformance between architecture and code.* Lack of conformance dooms an architecture to irrelevance as the code sets out on its own independent trajectory. We should work to find ways to establish conformance by construction (via generation, refinement, and augmentation), and by extraction (analyzing an artifact statically or dynamically to determine its architecture). Early work exists in both of these approaches, but we are a long way from reducing conformance to a solved problem, especially in recovery/enforcement of runtime views and architectural rules that go beyond structure.
- *Re-thinking our approach to software testing, based on software architecture.* An architecture can let us generate a wide variety of test plans, test cases, and other test artifacts. For code that originates in the architecture (such as implementations of connections and interaction mechanisms) automatic testing is possible. It should be possible to discriminate between code that originates in the architecture (such as that which implements connections and interaction mechanisms) and code that is non-architectural in nature (such as that which implements hidden functionality private to a component). We should

have different confidence levels in architectural versus non-architectural code, and we should be able to take advantage of that at test time. And it should also be possible to generate test plans, test cases, and other test artifacts from an architectural description of a system. A strong model of architecture-based testing, backed up by formal reasoning and easy-to-use tooling, could have a major economic impact on software system development.

- *Organizing architectural knowledge to create reference materials.* Mature engineering disciplines are characterized by handbooks and other reference materials that provide engineers with access to the systematic knowledge of the field. Cataloging architectural patterns [17] is a first step in this direction. But in addition, we need reference materials for analysis of realized architectures for evaluation of designs to predict properties of their implementation. Grady Booch's handbook on software architecture "codifying the architecture of a large collection of interesting software-intensive systems, presenting them in a manner that exposes their essential patterns and that permits comparisons across domains and architectural styles," [15] can provide important exemplars, but engineers also need reference material that organizes what we know about architecture into an accessible, dependable body of knowledge.
- *Developing architectural support for systems that dynamically adapt to changes in resources and each user's expectations and preferences.* As computing becomes ubiquitous and integrated in everyday devices, both base resources such as bandwidth and information resources such as location-specific data change dynamically. Moreover, each individual user has different needs that change with time. Developing architectures that can dynamically anticipate and react to these changes would help to maximize the benefit each user can obtain. Achieving this will require not only adaptive architectures but also component specifications that reflect variability in user needs as well as intrinsic properties of the component.

6. The golden age

It will be interesting to see how these ideas fare over the next ten years or – more likely – to see what ideas now undreamed of will have emerged. But one thing seems clear. The last decade and a half has seen a phenomenal growth of software architecture as a discipline. It started in the late 1980s as an academic idea, based on venerable foundations, that was aimed at understanding and codifying system descriptions observed in industrial practice. From there it has grown to a relatively mature engineering discipline complete with standard and repeatable practices, a rich catalog of pre-packaged design solutions, an enormous commercial market supplying tools and components, and a universal recognition that software architecture is an indispensable part of software system development.

A "golden age" is a period of prosperity and excellent achievement [6], often marked by numerous advances that rapidly move the technology from speculative to dependable. Consider, for example, the golden age of aviation: "Perhaps the most exciting years of aviation history span the period from the end of World War I to [the United States'] entry into World War II. This period is referred to as *golden* because of the countless advances in aviation technology that occurred, the many expeditions undertaken, and the numerous records set." [85] The last 15 years or so – roughly the middle four stages of the Redwine-Riddle model – truly have been the golden age for software architecture. Like the golden age of air travel in the 1930's, it has been an exciting time of discovery, unfettered imagination, great progress, great setbacks, and a sense of the possible.

But all golden ages come to a close, and as software architecture moves from being novel to being indispensable, its golden age is receding. This is as it should be. Because software architecture, like air travel after its golden age, is entering a period where it can be taken for granted. We rely on it, we cannot imagine our technological culture without it, and we are compelled to continually refine and improve it because it is indispensable.

The end of a golden age should not be taken to mean that the time for research, innovation, and improvement has passed. In aviation, enormous achievements such as jet engines, supersonic flight, pinpoint navigation, and space travel all happened well *after* its golden age had passed. So it will be with software architecture. The strong foundations laid by the early phases of software architecture maturation, coupled with ongoing research to make new ideas practical, will enable even more breathtaking system-building capabilities in the future. For us, the intriguing question is this: What new software engineering technology and *its* golden age will the solidly-established field of software architecture help to usher in?

7. Acknowledgements

Our thanks go to the program committee for the 2001 International Conference on Software Engineering for stimulating the original version of this paper. Thanks to Judy Stafford and Henk Obbink for commissioning this update for IEEE Software [79]. We appreciate invaluable

assistance from Sheila Rosenthal and Isaac Councill in helping us gather and analyze citation counts. Thanks to David Garlan and Jonathan Aldrich for helpful comments, and Dale Strok of *IEEE Software* for editing suggestions and for granting permission to use Figure 1. Mary Shaw's work is supported by the Software Industry Center, the AJ Perlis Chair of Computer Science, and the National Science Foundation under Grant CCF-0438929. The Software Engineering Institute is sponsored by the U.S. Department of Defense.

8. Appendix: Citation Analysis

To analyze the growth of the field, we analyzed citation patterns for books and papers with "software architecture" in the title. We obtained the results of a full search for such papers in the CiteSeer database [67]. We consolidated variant citations for papers and ignored self-citations, yielding a sample of about 5500 citations to about 750 books and papers. At the 2005 Working IEEE/IFIP Conference on Software Architecture (WICSA5), about 20% of the papers had "software architecture" in their titles. If this ratio holds for the literature at large, then these 750 represent about 20% of the software architecture papers.

This table lists the top two dozen books and papers in this sample, along with the top two dozen works in a similar (but unordered) sample from 2001. The table is ordered by 2005 rank, with membership in the 2001 set shown in the second column.

2005 rank	***2001 set***	***Pub year***	***Topic area***	***Authors***
1	yes	95	book	Shaw, Garlan. *SA: Perspectives on an Emerging Discipline[82]*
2	yes	96	book	Buschmann et al. *Pattern-Oriented SA vol 1 [17]*
3	yes	92	survey, model	Perry, Wolf. Foundations for the study of SA [68]
4	yes	93	survey, model	Garlan, Shaw. An introduction to SA [34]
5	yes	95	ADL	Shaw et al. Abstractions for SA and tools to support them [80]
6	yes	98	book	Bass, Clements, Kazman. *SA in Practice [12]*
7		94	formaliz ation	Magee et al. Specifying distributed SAs [54]
8	yes	94	specific domains	Macedonia et al. NPSNET: A Network SA for Large Scale Virtual Environments [53]
9		96	formaliz ation	Magee, Kramer. Dynamic structure in SAs [55]
10		92	formaliz ation	Allen, Garlan. A formal approach to SA [3]
11		95	formaliz ation	Inverardi, Wolf. Formal specification and analysis of SA [43]
12	yes	95	survey, model	Garlan, Perry. Intro to the Special Issue on SA [33]
13		98	specific domain	Decasper et al. Router Plugins: a SA for next generation routers [24]
14	yes	93	formaliz ation	Abowd, Allen, Garlan. Using style to understand descriptions of SA [1]
15		97	survey, model	Medvedovic, Taylor. A classification and compari-son framework for SA description languages [57]
16	yes	95	formaliz ation	Abowd, Allen, Garlan. Formalizing style to understand descriptions of SA [2]
17		94	analysis tech	Kazman et al. SAAM: a method for analyzing the properties of SAs [46]
18	yes	92	specific domains	Locke. SA for hard real-time applications [51]
19		98	specific domain	Frigo, Johnson. FFTW: an adaptive SA for the FFT [27]
20	yes	91	specific domains	Chiola: GreatSPN 1.5 SA [19]
21	yes	95	book	Walden, Nerson. *Seamless Object-Oriented SA [94]*
22		94	formaliz ation	Moriconi, Qian. Correctness and composition of SAs [62]
23	yes	94	specific domains	Chapman et al. A SA for multidisciplinary applications [18]
24		00	book	Schmidt et al. *Pattern-oriented SA vol 2 [72]*
25	yes	92	survey, model	Mettala, Graham. The domain-specific SA program [60]
29	yes	94	survey, model	Shaw, Garlan. Characteristics of higher-level languages for SA [81]
30	yes	96	formaliz ation	Le Metayer. SA styles as graph grammars [49]
35	yes	95	survey, model	Shaw, Garlan. Formulations and formalisms in SA [83]
36	yes	90	specific domains	Leung et al. A SA for Workstations supporting multimedia conferencing in packet switching Networks [50]
38	yes	97	rev eng	Yeh, Harris, Chase. Manipulating recovered SA views [97]

2005 rank	*2001 set*	*Pub year*	*Topic area*	*Authors*
47	yes	97	survey, model	Garlan. Research directions in SA [28]
54	yes	93	specific domains	Cremer et al. The SA for scenario control in the Iowa driving simulator [23]
92	yes	95	specific domains	Kruchten. The 4+1 view model of software architecture [48]
125	yes	92	specific domains	Coglianese, Goodwin, Kushner, Domain analysis for the avionics domain [22]

9. References

[1] Gregory Abowd, Robert Allen, David Garlan. Using Style to Understand Descriptions of Software Architecture, *Proc. 1st ACM SIGSOFT Symposium on the Foundations of Software Engineering*, December 1993

[2] Gregory Abowd, Robert Allen, and David Garlan. Formalizing style to understand descriptions of software architecture. *ACM Tr on Software Engineering and Methodology*, 1995.

[3] Robert Allen and David Garlan. A formal approach to software architectures. *Proc IFIP'92*, Elsevier, September 1992, pp. 134-141.

[4] Robert Allen and David Garlan. Formalizing architectural connection. *Proc 16th International Conference on Software Engineering*, May 1994, pp. 71-80.

[5] Robert Allen, David Garlan, and James Ivers. Formal modeling and analysis of the HLA component integration standard. Proc 6th Intl Symposium on the Foundations of Software Engineering, FSE-6, November 1998.

[6] Christine Ammer. The American Heritage® Dictionary of Idioms. Houghton Mifflin Company, 1997.

[7] Gregory Andrews. Paradigms for process interaction in distributed programs. *ACM Computing Surveys*, vol 23 no 1, March 1991, pp. 49-90.

[8] AT&T. "Best Current Practices: Software Architecture Validation." Internal report. Copyright 1993, AT&T.

[9] Felix Bachmann, Len Bass, and Mark Klein. Deriving Architectural Tactics: A Step Toward Methodical Architectural Design (CMU/SEI-2003-TR-004).

[10] Felix Bachmann, Len Bass, and Mark Klein. Preliminary Design of ArchE: A Software Architecture Design Assistant (CMU/SEI-2003-TR-021).

[11] Mario R. Barbacci, Mark H. Klein, and Charles B. Weinstock. Principles for Evaluating the Quality Attributes of a Software Architecture (CMU/SEI-96-TR-036), 1996.

[12] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley, 1998. Second edition, 2003.

[13] P. Blinn and S. Vestal. Formal real-time architecture specification and analysis. *10th IEEE Workshop on Real-Time Operating Systems and Software*, May 1993.

[14] Grady Booch. *UML Users Guide*. Addison Wesley, Chicago, Il, 1998.

[15] Grady Booch, *Handbook of Software Architecture* http://www.booch.com/architecture/index.jsp (accessed October 2005).

[16] Bredemeyer Consulting. *Training for Software Architects, System Architects and Enterprise Architects*. (Accessed October 2005.) http://bredemeyer.com/training.htm.

[17] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture - A System of Patterns*. Wiley and Sons, 1996

[18] B. Chapman, P. Mehrotra,, J. Van Rosendale, and H. Zima. A software architecture for multidisciplinary applications: Integrating task and data parallelism. *Tech. Rep. 94-18, ICASE*, NASA Langley Research Center, Hampton, VA, Mar. 1994.

[19] G. Chiola: GreatSPN 1.5 Software Architecture; *Proc. 5th Int. Conf. on Modeling Techniques and Tools for Computer Performance Evaluation*, Torino, 13-15 Feb. 1991.

[20] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Nord, J. Stafford: *Documenting Software Architectures: Views and Beyond*, Addison Wesley, 2002.

[21] P. Clements, R. Kazman, and M. Klein: *Evaluating Software Architectures: Methods and Case Studies*, Addison Wesley, 2001.

[22] Louis Coglianese, Mark Goodwin, Marty Kushner. Domain Analysis for the Avionics Domain Application Generation Environment of the Domain-Specific Software Architecture Project, *ADAGE-IBM-92-11*, Version 2.0, November, 1993

[23] J. Cremer, J. Kearney, Y. Papelis, and R. Romano. The software architecture for scenario control in the Iowa driving simulator. *Proc Conf on Computer Generated Forces and Behavioral Representation*, 1993.

[24] D. Decasper, Z. Dittia, G. Parulkar, B. Plattner. Router Plugins: A Software Architecture for Next Generation Routers In. *Sigcomm*, 1998.

[25] Norman Delisle and David Garlan. Formally specifying electronic instruments. *Proc. Fifth International Workshop on Software Specification and Design*, May 1989.

[26] E. W. Dijkstra. The structure of the THE multiprogramming system. *Comm ACM*, 1968.

[27] Matteo Frigo and Steven G. Johnson. FFTW: An adaptive software architecture for the FFT. *Proc IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 3, pages 1381-1384, May 1998.

[28] D. Garlan. Research directions in software architecture. *ACM Computing Surveys*, vol 27, no 2, pp. 257-261, 1995

[29] David Garlan. Software architecture: a roadmap. *The Future of Software Engineering 2000, Proceedings 22nd International Conference on Software Engineering*, ACM Press 2000.

[30] David Garlan, Robert Allen, and John Ockerbloom. Exploiting style in architectural design environments. *Proc SIGSOFT '94: 2nd ACM SIGSOFT Symposium on the Foundations of Software Engineering*, December 1994, pp. 170-185.

[31] David Garlan, Robert Monroe, and David Wile. Acme: An architecture description interchange language. *Proc CASCON '97*, November 1997, pp.169-183.

[32] David Garlan, Frances Paulisch, and Walter Tichy. Software Architectures, Report of the Dagstuhl Seminar 9508. Dagstuhl-Seminar-Report No 105, Internationales Begegnungs- und Forschungszentrum (IBFI), Schloss Dagstuhl, Germany, 1995.

[33] David Garlan and Dewayne Perry. Introduction to the Special Issue on Software Architecture, *IEEE Tr on Software Engineering*, vol 21 no 4, April 1995, pp 269-274.

[34] David Garlan and Mary Shaw. An introduction to software architecture. In *Advances in Software Engineering and Knowledge Engineering*, volume 1. World Scientific Publishing Co., 1993.

[35] Mark Goodwin and Marty Kushner, Domain Analysis for the Avionics Domain Architecture Generation Environment of Domain Specific Software Architecture. *ADAGE*-IBM-92-11.

[36] C. Hofmeister, R. Nord, and D. Soni. *Applied software architecture*. Addison-Wesley, 1999

[37] iCMG. Architecture Lab: Training. (Accessed October 2005).http://www.icmgworld.com/corp/Developer/dev.courseoverview.asp

[38] IEEE Computer Society Professional Practices Committee.. *Guide to the Software Engineering Body of Knowledge – SWEBOK. 2004*. (accessed October 2005) http://www.swebok.org/

[39] IEEE-Std-1471-2000, *Recommended Practice for Architectural Description of Software-Intensive Systems. IEEE, 2000*.

[40] IASA: International Association of Software Architects (accessed March 2006) http://www.iasahome.org

[41] IFIP WG 2.10: International Federation of Information Processing Societies Working Group 2.10 on Software Architecture (accessed March 2006) http://www.softwarearchitectureportal.org/

[42] International Software Quality Institute. *Training for Professional for Software Architecture.* (Accessed October 2005). http://www.isqi.org/isqi/eng/cert/ca/

[43] P. Inverardi and A. Wolf. Formal Specification and Analysis of Software Architectures Using the Chemical Abstract Machine Model. IEEE Transactions on Software Engineering, vol 21, no 4, 1995, 373--386.

[44] Joint Task Force on Computing Curricula, IEEE Computer Society and ACM. Software Engineering 2004, *Guidelines for Undergraduate. Degree Programs in Software Engineering*. August 2004. (accessed October 2005) http://sites.computer.org/ccse/SE2004Volume.pdf

[45] R. Kazman, M. Barbacci, M. Klein, S.J. Carrière, Experience with Performing Architecture Tradeoff Analysis, *Proc ICSE '99*, May 1999, 54-63.

[46] R. Kazman, L. Bass, G. Abowd and M. Webb. SAAM: A method for analysing the properties of software architectures, *Proc. 16th Int. Conf. Software Engineering*, pp. 81-90, 1994.

[47] R. Kazman, P. Clements, L. Bass, and G. Abowd.. Classifying Architectural Elements as a Foundation for Mechanism Matching, Proc. *COMPSAC '97 International Computer Software and Applications Conference*, August 1997, pp. 1417.

[48] P. Kruchten. The 4+1 View Model of Software Architecture. *IEEE Software* (Nov. 1995): 42-50.

[49] D. Le Metayer. Software architecture styles as graph grammars. In Proc ACM SIGSOFT '96 4th Symposium on the Foundations of Software Engineering, 1996, pp. 15-23.

[50] Wu-Hon F. Leung, Thomas J. Baumgartner, Yeou H. Hwang, Mike J Morgan, ShiChuan Tu, A Software Architecture for Workstations Supporting Multimedia Conferencing in Packet Switching Networks, *IEEE Journal on Selected Areas in Communications*, vol 8, no 3, pp. 380-390, April 1990

[51] C. Locke. Software architecture for hard real-time applications: Cyclic executives vs. fixed priority executives. *Journal of Real-Time Systems*, vol 4 no 1, March 1992, pp. 37-53.

[52] D.C. Luckham, L.M. Augustin, J.J. Kenny, J. Veera, D. Bryan, W. Mann. Specification and analysis of system architecture using Rapide. *IEEE Tr on Software Engineering* vol 21 no 4, April 1995, pp 336-355.

[53] Michael R. Macedonia, Michael J. Zyda, David R. Pratt, Paul T. Barham, and Steven Zeswitz, NPSNET: A Network Software Architecture for Large Scale Virtual Environments. *Presence, Teleoperators, and Virtual Environments*, vol. 3, no. 4. Fall 1994.

[54] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. Specifying distributed software architectures. *Proc 5th European Software Engineering Conference*, September 1994.

[55] J. Magee and J. Kramer: Dynamic Structure in software architectures. *Proc. ACM SIFSOFT'96: Fourth Symposium on the foundations of software engineering* (FSE4), pp. 3-14, ACM Press 1996. Online reference: http://portal.acm.org/citation.cfm?id=239104

[56] N. Medvidovic et al. Using Object-Oriented Typing to Support Architectural Design in the C2 Style. In *Proceedings of the ACM SIGSOFT '96 Fourth Symposium on the Foundations of Software Engineering*. p.24-32, ACM SIGSOFT. San Francisco, CA, October 1996.

[57] N. Medvedovic, R.N. Taylor, A Classification and Comparison Framework for Software Architecture Description Languages. Tech. Report UCI-ICS-97-02, Department of Information and Computer Science, University of California, Irvine, Feb. 1997.

[58] N. Medvidovic and R.N. Taylor. A Framework for Classifying and Comparing Architecture Description Languages. *Proc 6th European Software Engineering Conference*, Lecture Notes in Computer Science 1301, pages 60--76, September 1997.

[59] N. R. Mehta, N. Medvidovic, and S. Phadke. Towards a taxonomy of software connectors. *Proc. International Conference on Software Engineering, 2000.*

[60] E. Mettala and M. Graham (eds.), The Domain-Specific Software Architecture Program, *Technical Report CMU/SEI-92-SR-9*, Software Engineering Institute, Carnegie Mellon University, 1992.

[61] Microsoft .NET home (accessed March 2006) http://www.microsoft.com/net/basics.mspx

[62] M. Moriconi and X. Qian. Correctness and Composition of Software Architectures. *Proc Second ACM SIGSOFT Symposium on Foundations of Software Engineering*, Software Engineering Notes, December 1994

[63] H. Penny Nii. Blackboard Systems. *AI Magazine* vol 7, no 3, 1986, pp:38-53 and vol 7, no 4, 1986, pp. 82-107.

[64] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, vol. 15, no. 12, December 1972, pp. 1053-1058.

[65] D. L. Parnas. On a "buzzword": hierarchical structure. *Proc IFIP Congress 74*, 1974.

[66] D.L. Parnas, P.C. Clements, and D.M. Weiss. The Modular Structure of Complex Systems. *IEEE Tr on Software Engineering*, Vol. SE-11, No. 3, March 1985, pp. 259-266.

[67] Penn State School of Information Sciences and Technology. *CiteSeer.IST Scientific Literature Digital Library* (formerly ResearchIndex). Public online search engine and digital library. (accessed October 2005) http://citeseer.ist.psu.edu/

[68] D. E. Perry and A. L. Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 17:pp. 40-52, October 1992.

[69] Raytheon Company. *Raytheon Enterprise Architecture Process* (accessed October 2005). http://wwwxt.raytheon.com/technology_today/v3_i2/feature_ent_arch.html, 2004

[70] Roger Pressman. *Software Engineering: A Practitioner's Approach.* McGraw Hill 2004.

[71] Samuel Redwine and William Riddle. Software technology maturation. *Proc 8th International Conference on Software Engineering*, May 1985, pp. 189-200.

[72] D. Schmidt, M. Stal, H. Rohnert, F. Buschmann. *Pattern-Oriented Software Architecture, Vol. 2: Patterns for Concurrent and Networked Objects.* Wiley 2000.

[73] Selby, Richard W. & Reimer, Ronald M. Interconnectivity Analysis for Large Software Systems. *Proceedings of the California Software Symposium.* California. (March 1995): 3-17

[74] Mary Shaw. Toward Higher-Level Abstractions for Software Systems. *Proc. Tercer Simposio Internacional del Conocimiento y su Ingerieria*, October 1988 (printed by Rank Xerox) (invited), pp.55-61. Reprinted in *Data & Knowledge Engineering*, vol. 5, no 2, July 1990, pp. 119-128 Elsevier Science Publisher 1990, online reference http://portal.acm.org/citation.cfm?id=87367. Revised as Larger-Scale Systems Require Higher-Level Abstractions, *Proc. 5th Int'l Workshop on Software Specification and Design*, Pittsburgh, May 1989, pp.143-146.

[75] Mary Shaw. Elements of a Design Language for Software Architecture, Position Paper for *IEEE Design Automation Workshop*, January 1990.

[76] Mary Shaw. Heterogeneous Design Idioms for Software Architecture, *Proc. Sixth International Workshop on Software Specification and Design*, pp. 158-165, IEEE Press, October 1991.

[77] Mary Shaw. The Coming-of-Age of Software Architecture Research. *Proc 23rd International Conference on Software Engineering*, 2001, pp. 656-664a.

[78] Mary Shaw and Paul Clements. A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems. *COMPSAC '97 International Computer Software and Applications Conference*, August 1997, pp 6-13.

[79] Mary Shaw and Paul Clements. The golden age of software architecture. *IEEE Software*, vol 23, no 2, March/April 2006, pp. 31-19.

[80] M. Shaw, R. DeLine, V. Klein, T.L. Ross, D.M. Young, G. Zelesnik. Abstractions for Software Architecture and Tools to Support Them. *IEEE Tr on Software Engineering*, vol. 21, no 4, April 95.

[81] Mary Shaw and David Garlan. Characteristics of Higher-level Languages for Software Architecture. *Carnegie Mellon University Computer Science Technical Report CMU-CS-94-210*, December 1994.

[82] Mary Shaw and David Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[83] Mary Shaw and David Garlan. Formulations and formalisms in software architecture. Invited for special volume of Lecture Notes in Computer Science, *Computer Science Today: Recent Trends and Developments*, Jan van Leeuwen (Ed) Springer-Verlag 1996, pp.307-323.

[84] C. Smith, Performance Engineering,. 794-810. *Encyclopedia of Software Engineering*, Vol. 2. New York, NY: Wiley, 1994

[85] Smithsonian National Air and Space Museum. *The Golden Age of Flight.* (accessed March 2006) http://www.nasm.si.edu/exhibitions/gal105/gal105.html

[86] Society for Automotive Engineers, *Architecture Analysis & Design Language (Aadl)*, SAE Standard AS5506, November 2004, (accessed December 2005) http://www.sae.org/servlets/productDetail?PROD_TYP=STD&PROD_CD=AS5506

[87] Software Engineering Institute. *How Do You Define Software Architecture?* (accessed October 2005) .http://www.sei.cmu.edu/architecture/definitions.html

[88] Software Engineering Institute. *Software Architecture Curriculum and Certificate Programs* (accessed October

2005).
http://www.sei.cmu.edu/architecture/arch_curriculum.html

[89] Software Engineering Institute. *Upcoming Events in Software Architecture?* (accessed December 2005) .http://www.sei.cmu.edu/architecture/events.html

[90] D. Soni, R. Nord, and C. Hofmeister. Software Architecture in Industrial Applications, *Proc. Seventeenth Intl. Conf. Software Engineering*, pp. 196-207, ACM Press, 1995.

[91] Kevin Sullivan, M. Marchukov and D. Socha. Analysis of a conflict between interface negotiation and aggregation in Microsoft's component object model. *IEEE Trans on Software Engineering*, July/August, 1999.

[92] Clemens Szyperski. *Component Software – Beyond Object-Oriented Programming*, Addison Wesley, 1997.

[93] Hans van Vliet. *Software Engineering: Principles and Practice*, 2nd Edition, Wiley, Sept 2000.

[94] Kim Walden and Jean-Marc Nerson. *Seamless Object-Oriented Software Architecture - Analysis and Design of Reliable Systems*. Prentice Hall, 1995.

[95] WICSA: The Working IEEE/IFIP Conference on Software Architecture (accessed March 2006) http://www.softwarearchitectureportal.org/WICSA/conferences/index.htm

[96] WWISA: The Worldwide Institute of Software Architecture (accessed March 2006) http://www.wwisa.org

[97] A. Yeh, D. Harris, and M. Chase. Manipulating recovered software architecture views. Proc 19th Int'l Conf on Software Engineering, 1997