Species evolve, individuals age

Invited Keynote Talk

Mehdi Jazayeri
Faculty of Informatics, University of Lugano
and
Distributed Systems Group, Technical University of Vienna

Abstract

The phenomenon of "software evolution" was observed back in the 1970s when the first large software systems were being developed, and it has attracted renewed attention in the 1990s. Yet, the difficulties of software evolution are still treated as problems that pop up unexpectedly in software projects. In this talk, I look at some possible reasons to explain this apparent contradiction and the possibly related confusion in software evolution research. I point out some ways we can make software evolution research more relevant to software engineering practice. My primary goal is to clarify that what evolves is not the software but our knowledge about a particular type of software.

1. Introduction

Software evolution is now a common phrase and an accepted research area in software engineering. There are conferences and workshops, such as this one, devoted to the topic. It is also a common key phrase in general software engineering papers. The 2004 ACM/IEEE Software Engineering Curriculum Guidelines list software evolution as one of ten key areas of software engineering education. And there are several research groups and international networks working on software evolution. And, as perhaps may be expected, there are diverging research efforts in sub-areas of software evolution, spanning theoretical studies, empirical studies, tools, visualization, and so on. With all this effort going into researching software evolution, it is reasonable to ask if we are working towards a common goal and how we can relate the diverse efforts to one another.

There are interesting fundamental questions that make software engineering research difficult. Such questions also plague software evolution. There have been debates about the term "software engineering" since the early days and heated discussions about whether it is engineering at all. We can pose the same questions about software evolution. Is evolution the right word to use? If it is, is software evolution the right term? Is it really software that evolves? Words are important because they shape our thinking and approach to a problem. If we don't name the problem right, we will find solutions to the wrong problem.

In this paper, I will take a broad look at the efforts on software evolution research and offer some opinions that I hope identify some problems we must face and propose some directions worth exploring.

2. Does software evolve?

Since the classic and insightful work of Lehman and Belady [Lehman85], we have all accepted "software evolution" as a phenomenon worth studying and one that we acknowledge poses serious problems to software projects. But is "software evolution" the right name for the problem we should be attacking?

In this section, I will examine the term "software evolution" more closely. Let us start with the word "software", move on to the word "evolution", and then putting the two together, we can ask if it is really "evolution of software" we are talking about.

While we all know what software is, it is surprisingly difficult to come up with a complete definition of software. Without getting into philosophical discussions, we can agree that software is certainly more than the bits stored in a file, or even the source code stored in another file. When we refer to software we generally mean a more abstract idea that encompasses the concepts and algorithms embodied in the implementation. Sometimes, to be more precise, we say the software and all its associated artifacts such as documentation of its architecture, design, test data, requirements, etc. Osterweil famously remarked that "processes are software too" and in [Osterweil03] discusses the nature of software and laments the fact that we know so little about it. He correctly points out that we generally define software by what it is not. It

"is not tangible, it is not physical, and it cannot be accessed by any of our senses." It is quite amazing that something we cannot define precisely is responsible for so many technological advances. Unfortunately, the inability to come to grips with a proper definition of software clouds most of our research in the software area, and the area of software evolution is no exception.

Evolution in general parlance implies that something has changed for the better. The Merriam-Webster Dictionary defines evolution as "a process of continuous change from a lower, simpler, or worse to a higher, more complex, or better state," which captures our intuitive notion of something improving. Of course, we associate the term mostly with biology. The same dictionary gives a more scientific definition in this context: "a theory that the various types of animals and plants have their origin in other preexisting types and that the distinguishable differences are due to modifications in successive generations." Indeed, here is the crucial point of evolution: changes occur in succeeding generations. An individual animal or plant does not evolve; rather, its species evolves over time. And typically the evolution is for the "better" because it enables the species to cope better with the environment.

So what do we mean when we talk about "software evolution"? We rarely talk about generations of software. We typically concentrate on one software product, possibly in relation to other information about the product. For example, we examine successive releases of the source code, or examine change data stored in source code control systems. Our studies almost always trace the source code or some information about the source code. Thus, in contrast with the biological definition, we trace the "evolution" of an individual, rather than a species.

But is there something analogous to "species" when we talk about software? The answer is definitely yes. The species are the high-level models that we use to describe (aspects of) software. An architectural description, in fact, describes a whole species of software systems. The family architecture approach to software development makes this explicit by capturing a whole family (species) of systems in terms of their commonalities and differences. If evolution does take place in software, we can hope that it occurs at these meta-levels, where new architectures are created as improvements to previous architectures, leading to evolved species. Individual elements in the family certainly change over time but this change is hardly evolutionary in the sense that it leads to long-term improvement. What we do know about software and even Lehman's laws of evolution is that any individual software system will eventually reach an old age when it is no longer cost-effective to modify it and it is better to retire it.

On the other hand, another classic paper, this one by Parnas [Parnas94] talks about "software aging," which is a much more appropriate description of what happens to source code. As [Eich01] clearly demonstrates, individual software ages. What I have tried to argue is that software species as a whole evolve—that is, we find better models and architectures—while individual systems decay and die. An interesting exception to this paradigm is the class of evolving software, which is modified (we can even say evolved) while running. A great example is the Internet software, which has evolved through generations as it keeps providing more and more services.

3. Evolution is in the mind

If it is not the source code that evolves, then what does evolve? Clearly, our ability to produce software improves and we can at least intuitively identify generations of software systems that each are improvements over previous ones. These new generations typically have better structure and are based on newer technologies that reinforce and sometimes enable the improved structure. We have better understanding of the requirements and we have better modeling and design methods. All these improvements enable us to build a better generation of the same software system. Sometimes we may use some of the source code from the previous generation but that is an incidental point. The evolution happens at a higher level.

In a general sense, what evolves is our collective knowledge about a particular kind of software system. But this collective knowledge is hard to observe and measure. So, we look for observable evidence of evolution. Of course some of the artifacts change and we can observe and measure those changes and we can make all kinds of inferences and generalizations from those observations and hypothesize about the evolution of those artifacts. But those artifacts change in response to external factors that we usually cannot measure and observe objectively. As a result, any measurements give us only an incomplete picture of the true evolutionary phenomena. This is a Heisenberg-like principle in observations of software evolution. We are guaranteed that any measurements we make are an incomplete picture of the phenomenon we want to observe. Another uncertainty principle is that as we collect data on the aging of a particular piece of software, our knowledge about how to build that kind of software evolves based on experience with that type

of software and also based on evolution of other factors such as new technologies. Therefore, any knowledge we gain from such measurements apply only to old times and not to the current state of affairs. Worse, we can even not know how relevant our conclusions about the measurements are because we cannot relate our measurements to evolutions of other factors.

4. Evolution of a web site: a case study

Since 1995, the Distributed Systems Group at the Technical University of Vienna has been developing the Website for the Vienna International Festival (www.festwochen.at), an annual cultural event, possibly the cultural highlight of the summer in Vienna. The festival hosts different types of events—plays, concerts, movies, musicals—and visitors come from all over the world. Even back in 1995, in the early days of the Web, a Website seemed an ideal advertising venue for the Vienna International Festival. My group was looking for a real-world case study to experiment with Web development. The project was a good fit for us because the festival organization was adventurous and wanted us to be innovative. Since then, we have developed the Website with different kinds of technologies and invented some new techniques to make the Website development more manageable. The evolution of the Website presents a good case study of the forces that influence evolution. The site currently consists of HTML, XML, PHP, Java, Javascript, and other codes. It has grown considerably in both power and complexity since its beginning. To cope with the complexity, we have developed supporting technologies that are used in the development of the Website.

The Website started in 1995 as purely a repository of information. In those days, using the table feature of HTML was a challenge! The users could simply view information about the offered events. In effect, it was a calendar of events along with pricing information. There was a phone number which users could call to reserve tickets. More sophisticated users could send us (at the Distributed Systems Group) an email with their request, including their credit card numbers, and we would transmit this information to the Vienna International Festival office. Even for this simple site, it was clear that pure HTML was insufficient. The driving force at that time was that we wanted every page to have the same look-and-feel—the so-called corporate identity. With pure HTML, any small changes to look-and-feel would have required changes to all the pages. To cope with this problem, Robert Barta developed a pre-processor language called HTML++ [Barta95], which supported the factoring of information that was common to many pages. This marked the beginning of a series of work we carried out in "Web engineering." It was clear that writing HTML code for complicated Websites was not a real option. We continued the generative approach set by HTML++, and created increasingly more powerful languages for generating the Website. Thus, we evolved a set of a techniques or languages for generating the Website. Of course, looking at the Website from year to year one can see the changes of the Website but the real evolution happened on the tool side.

During 1996-1998, Markus Schranz created the Jessica Web engineering system [Barta98], which provided object-oriented mechanisms for writing hypermedia documents. The code of the Vienna International Festival began to contain classes and objects. For example, all pages inherited a class that described the corporate identity. During these years, two other important developments occurred: on-line purchasing of tickets became acceptable and even popular, and the graphical look of the Website took on more importance. These two phenomena added considerably to the complexity of the Website. For one thing, an external design firm was hired to create the graphical design. They naturally cared more about the appearance of the content on the screen than the implementation effort required to support the interaction with the user. Supporting on-line purchase required interfacing with a financial clearing-house, with very strict security and functional protocols. These complexities led to a substantial increase in the amount of Java code in the implementation.

To cope with the new complexities, the group invented the MyXML tool [Kirda00a, Kirda00b], to support the separation of the layout, the content, and the business logic of a Website. This approach directly addressed the two problems we faced. The graphic design was provided by a separate group; the required programming to support the "business logic", such as a shopping cart, had become a major subsystem on its own; we could now develop these two components independently and integrate them with the help of MyXML. To further enhance the separation of the content from the layout and the logic, the group also developed a content-management system to allow the festival employees to enter the data directly into the system [Kerer02b].

In parallel with the development of these tools, we were also examining and experimenting with new developments in Web languages and software engineering design methods. Schranz [Schranz99] explored the use of UML for designing and documenting the structure of the Website. Our goal

was to be able to preserve an architecture for the Website that persisted from year to year. Kerer [Kerer02a] developed a methodology for implementing separation of concerns in Web engineering by using the software engineering concept of contracts for specifying the modules of a Website and their responsibilities and expectations. We also explored the use of XSL [Kerer01]. These design techniques influenced how we went about the design of the Website and therefore changed the implementation code of the site. But the relationship between one release of the Website to the next can only be traced through the evolution of our design methodology.

Another factor that influenced the evolution of the Website was the constant development of new technologies. For example, with the advent of diverse web-enabled devices, we needed to be able to present the festival on such devices. Kirda [Kirda01b, Kirda03] developed an approach for designing Websites that are relatively independent of the presentation platform. The latest evolution, currently in progress, is the integration of semantic web concepts into the Website. Gerald Reif [Reif05] has developed a technique for including semantic information (using RDF) into the Website so that programs such as software agents may also be able to query the Website about events and availability of tickets. Another planned work is the migration of the Website to the Cocoon web development framework (www.cocoon.apache.org), an open-source tool which is more complete than MyXML.

An interesting side note is that after the first year (1995) we realized that we were developing something more than a Website. We were in fact building some sort of application to interact with the user. We considered calling it a Web application but settled on calling it a Web service and we called our trade "web service engineering." We did not know that later on the term "web service" would be co-opted to mean something different. With the inclusion of semantic content in the Vienna International Festival Website, however, the site is perhaps approaching the level of offering services.

Reviewing the life of the Website over the last ten years, we can clearly see that it has evolved in the sense that each year it has offered more features, and it has been better structured, according to advances in Web engineering methodologies. We can classify the factors that contributed to the evolution of the Website into two categories:

1. Need for new features. The first version offered static text and image access. Competition from other Websites and new Web technologies made it possible to include many additional capabilities including shopping-cart, interest forums, area devoted to the press, advertising, and archives of previous years.
2. Web engineering technologies. The new technologies enabled better design and architecture, and therefore more powerful Websites. In general, the new technologies addressed ever higher-level design issues. Starting with the HTML code as the assembly language, each new technology provided slightly better modeling capabilities, enabling the Web developer to design the structure of the Website. HTML++, XML, Jessica, XSL, MyXML each supported the evolution of the Website. The new technologies made it possible to meet the need for new features.

Considering the code that makes up the Website, we can ask several questions to clarify the notion of evolution:

1. What constitutes the Website? (a version of the question, "what is software?")
2. What has evolved over the years?
3. Is it the same Website this year as it was ten years ago?

In some general sense the Website is the same as it was ten years ago. It still interacts with the user and provides services in response to user requests about the Vienna International Festival. Except for the fact that the information about the festival is different, the site offers more features, the code that implements the site is totally different, and the machine on which the service runs is different, it is basically the same! Of course, the Website includes all the code that runs when the user uses a browser to interact with it. It consists of many lines of code in different languages, but also a few large databases that contain different kinds of content. It also consists of configuration information. But a lot of what we produce when we create the Website does not actually run when a browser calls the Website. This information includes the design of the Website and the various tools that are used to generate and test the Website. It is this information that has truly evolved over time. Indeed, every year or two, we have used a new generation of Website generation technology that has changed the design of the Website dramatically. The models we use—supported and enabled by the tools—have evolved. These models and tools reflect our current understanding of how to build good Websites.

In the case of this Website, we have had the luxury of starting from scratch each year. We carry with us the knowledge from the past year and (almost) none of the code. In this sense, the site evolves because each year is a different instance of the same Website but it is not the same (modified) Website.

Table 1 summarizes the evolution characteristics of the Website.

Year	Language	Feature	Input form
1995	HTML++	Static	Plain text
1996	HTML++	Off-line ticket purchase	
1997	Jessica	On-line tickets	
1998	Jessica		Oracle
1999	Jessica		MySQL
2000	XML/XSL	Shopping cart	
2001	XML/XSL	Ticketing outsourced	WebCUS Content Update System
2002	XML/XSL		
2003	MyXML	Video streaming	
2004	MyXML		
2005	MyXML	Semantic Web	

Table 1. Evolution of the festival Website

5. Where to look for solutions?

Having separated the notions of individuals and species, where can we look for fruitful approaches to addressing software evolution problems? I believe that we need to look at approaches that clearly distinguish between individuals and species and allow us to observe, measure, monitor, and reason at the species level. When dealing with software, such techniques are often based on modeling and meta-modeling. I propose two areas worth considering. Both are instances of approaches to general software engineering and I believe they also apply to software evolution.

5.1 Domain specific software evolution

As software engineers, we often fall into the trap of treating software as if there were a generic artifact called software and we could make general statements about it, its properties, and ways of constructing and manipulating it. Well, there are some generic things we can do with software, such as editing it, storing it in a file system, copying it, and so on. But these are not the interesting aspects of software. The most interesting aspects about software are not generic at all. In fact, the distinctions between different types of software are what make dealing with software difficult and interesting. We naturally accept that there are differences between system software and application software and embedded software. Software cost estimation methods, in fact, take such differences into account because the effort required to build a software product depends heavily on the type of the software product. Yet, too often we tend to forget these differences. We do this also in software evolution. Is there any reason why we should expect the evolution of a Web application to follow the same patterns and have the same requirements as that of word-processing software? I don't think so.

Instead of treating all software alike, we should look at the evolution of domain-specific software. Avoiding the difficulties of defining the term domain, I would use the terminology of this paper to say we should look for species-specific evolution. I mention three different types of species that appear to have different evolution characteristics.

1. Web applications and services. As the case study described, Web-based applications have specialized requirements and environments. They have specific types of stakeholders, specific types of rapidly changing technologies, require the integration of many different types of languages and tools, are often database-centered, are multi-tiered, and have to be highly scalable. These characteristics lead to evolution characteristics that are very different from generic software.
2. Open-source software. The availability of the complete code of open-source software projects makes it possible to analyze the evolution of individual software projects on the basis of the source code and other data such as bug reports. Various interesting analyses of CVS repositories have been reported and many interesting techniques have been developed by researchers. But it is important to keep in mind that open-source projects have different characteristics from other species of software. In this case the differences stem from, such factors as the kinds of processes used, the interaction patterns among team-members, and different reporting and collaboration structures. Indeed, Scacchi [Scacchi04] reports that the Lehman laws of evolution may not apply to open-source software. A study of open source evolution is reported in [Godfrey00].
3. Component-based software. Component-based software offers interesting challenges in the study of software evolution. First, we can combine different individual elements to build new species of software. Second, the individual elements may have their own evolution patterns, time-scales, and histories just as the species built from them begin their own evolution. This implies that we need tools and techniques to analyze the evolution of systems and their components in parallel, both independently from each other and

in relation to each other. Of course, different species of software may be themselves component-based. For example, web applications are almost always component-based. We have to be able to deal with this combinatorial relationship in the evolution patterns.

I have only mentioned three types of species that most likely have different types of evolution characteristics. There are many more. I believe that laws, theories, tools, and methods that address software evolution will be more successful if they are species-oriented and species-specific.

5.2 Family-oriented software evolution

As I have already mentioned, software architectures allow us to describe a species of software products and are thus an appropriate vehicle for the study of software evolution. Product-family approaches [Jazayeri00] raise the level of reasoning to a more abstract level and are therefore even more appropriate. Product families have proven useful for product development because they capture the commonalities of family members and deal with variations that distinguish individual members of the family. These same characteristics can help in evolution studies. Indeed, a family-oriented approach to software evolution would force us to capture the model elements that are usually ignored (or lost) when we study only the source code. More importantly, since most software is not developed with a family architecture, we must try to construct a family architecture and use it as a vehicle for studying the evolution of the software species. A family architecture is a description of the species and therefore a necessary starting point for evolution studies. Many reverse engineering techniques are useful in this regard and some even try to recover family architectures.

6. Summary

This paper has argued that the study of software evolution faces the same fundamental problems of software engineering, the most important one being the lack of a proper definition of software. The message of the paper may be summarized in three simple points:

1- Software is more than source code. Models (and meta-models) are more important aspects of software that play key roles in software evolution.
2- Individual software products age while our understanding of them and—as a result—their models (and meta-models) evolve.
3- Evolution studies must focus on the species of software rather than on individual software products. The distinction between the individual and the species is essential.

7. Acknowledgements

This work was supported in part by the ITEA Families project. The author is a member of the RELEASE network supported by the European Science Foundation.

8. References

[Barta95] Robert Barta, What the Heck is HTML++ – Salvation for the Souls of Webmasters, Technical Report TUV-1841-1995-06, Distributed Systems Group, Technical University of Vienna, (Available at http://www.infosys.tuwien.ac.at/).

[Barta98] Robert Barta and Markus W. Schranz. Jessica - an object-oriented hypermedia publishing processor. Computer Networks and ISDN Systems, 30(1–7):281, Apr. 1998.

[Eich01] Stephen G. Eick, Todd L. Graves, Alan F. Karr, J.S. Marron, Audris Mockus, “Does Code Decay? Assessing the Evidence from Change Management Data,” IEEE Transactions on Software Engineering, vol. 27, no. 1:1 – 12, January 2001.

[Gall97] Harald Gall, Mehdi Jazayeri, René Klösch, Georg Trausmuth: Software Evolution Observations Based on Product Release History. Proceedings of ICSM 1997: 160–169.

[Gîrba04] Tudor Gîrba, Stéphane Ducasse and Michele Lanza, “Yesterday's Weather: Guiding Early Reverse Engineering Efforts by Summarizing the Evolution of Changes,” Proceedings of ICSM '04, IEEE Computer Society Press, 2004, pp. 40–49.

[Godfrey00] Michael W. Godfrey and Qiang Tu, “Evolution in Open Source Software: A Case Study,” in Proc. of the 2000 Intl. Conference on Software Maintenance (ICSM-00), San Jose, California, October 2000.

[Jazayeri00] Mehdi Jazayeri, Alexander Ran, Frank Van der Linden. Software Architecture for Product Families: Principles and Practice, Addison-Wesley, 2000.

[Kerer01] Clemens Kerer, Engin Kirda, Mehdi Jazayeri, and Roman Kurmanowytsch. Building XML/XSL-Powered Web Sites: An Experience Report. In 25th International Computer Software and Applications Conference (COMPSAC), Chicago, IL, USA, Oct. 2001. IEEE Computer Society Press.

[Kerer02a] Clemens Kerer, Engin Kirda, and Christopher Kruegel. XGuide – a practical guide to XML-based web engineering. In Lucy Cherkasova and Fabio Panzieri, editors,

International Workshop on Web Engineering, Networking 2002, Pisa, Italy, May 2002, May 2002. Springer.-Verlag.

[Kerer02b] Clemens Kerer, Engin Kirda, and Roman Kurmanowytsch. A Generic Content-Management Tool for Web Databases. IEEE Internet Computing, 6(4):38-42, July/August 2002.

[Kirda00a] Engin Kirda and Clemens Kerer. MyXML: An XML-based template engine for the generation of flexible web content. In WEBNET 2000, San Antonio, Texas, USA, Nov. 2000.

[Kirda00b] Clemens Kerer and Engin Kirda. Layout, content and logic separation in web engineering. In 9th WWW. 3rd Web Engineering Workshop, Amsterdam, 15-16 May, 2000, Lecture Notes in Computer Science Series, 2016, Springer Verlag.

[Kirda01a] Engin Kirda, Mehdi Jazayeri, Clemens Kerer, Markus W. Schranz: Experiences in Engineering Flexible Web Services. IEEE MultiMedia 8(1): 58-65, Jan.–Mar. 2001.

[Kirda01b] Engin Kirda. Web engineering device independent web sevices. In 23rd ICSE., Doctoral Symposium, Toronto, Canada, May. 2001. IEEE Computer Society Press.

[Kirda03] Engin Kirda, Clemens Kerer, Christopher Kruegel, and Roman Kurmanowytsch. Web Service Engineering with DIWE. In 29th EUROMICRO Conference, Antalya, Turkey. IEEE Computer Society Press, September 2003.

[Lehman85] Meir M. Lehman and L.A. Belady, Program Evolution – Processes of Software Change, Academic Press, London, 1985.

[Osterweil03] Leon J. Osterweil, Understanding process and the quest for deeper questions in software engineering research, Proceedings of the 9th European Software Engineering Conference held jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC-FSE 2003): 6–14, Helsinki, Finland, 2003.

[Parnas94] David L. Parnas, “Software Aging,” Proceedings of the 16th International Conference on Software Engineering (ICSE 1994), pp. 279 – 287, Sorrento, Italy, 1994.

[Reif05] Gerald Reif, Harald Gall, Mehdi Jazayeri. WEESA–Web Engineering for Semantic Web Applications. 14th International World Wide Web Conference (WWW), pp. 722-729, Chiba, Japan, 2005.

[Scacchi04] Walt Scacchi, “Understanding Open Source Software Evolution: Applying, Breaking, and Rethinking the Laws of Software Evolution,” July 2004.

[Schranz97] Markus Schranz. Vienna international festival (Wiener festwochen) 1996 - managing culture on the WWW. In Museums on the WEB., Los Angeles, California, Mar. 1997.

[Schranz99] Markus Schranz, Johannes Weidl, and S. Zechmeister. Using UML for the engineering of complex worldwide web services. In 8th WWW. (WWW '99), Toronto, Canada, May 1999.