On The Issue of Architecture-Preserving Methodologies and Their Identification[1]
Rev2.0[2]

Karl Reed [3]
kreed@cs.latrobe.edu.au
School of Information Technology
Bond University
Gold Coast QLD 4229 Australia

Jason Baragry
jason.baragry@nr.no
Norsk Regnesentral,
Postboks 114 Blindern, N-0314 OSLO.
Norway

Abstract

One of the fundamental arguments for the concept of Software Architecture is that it will permit designers to identify an "architecture" early in the design, and to use it to guide implementation. However, there is a significant amount of evidence that this goal is not always realised. This is shown by work in the re-engineering community which examines the relationship between the presumed architecture of large software systems, and that which can be identified from the finished code. It is also the centre of a conjecture by one of the authors that there are circumstances where this cannot be achieved. This paper makes the case for assessing implementation methodologies against the criteria of "architecture preservation". It argues that empirical studies should be made to test the hypothesis that methodology has (or has no) influence upon the extent to which a priori architectures are preserved in a completed system. In presenting this argument, factors which will complicate the outcomes, for example, poorly chosen a priori architectures, are noted.

1. Introduction

The idea that software or systems have an "architecture" seems to have arisen very early in our discipline. Baragry**[1]** has traced the use for this term back to the 1960's, (see Brooks **[7]**) and there is reason to believe it may have been used even earlier. As is well known, the importance of "architecture" (or rather, design integrity) was raised forcibly by Brooks **[8]**, and was brought to prominence by Shaw in the mid 1980's (Shaw **[23]**). Subsequently, the field has developed, and has reached a level of maturity sufficient to support several conferences and numerous workshops.

However, there remain a number of difficulties with both concepts in the field and their interpretation, as evidenced by the work of a number of researchers. Despite this, the over-arching concept of "architecture" for a software system is of great importance. It is certainly the case that it is believed that there do exist standard

---

[1] Accepted for publication as a Position paper, ICSE 2002 Workshop on SARA Software Architecture Review and Assessment May 19 2002 Orlando
[2] Rev 2.0 23/4/2006 added footnote re SARA workshop.
[3] On leave from the Department of Computer Science and Computer Engineering, La Trobe University, Bundoora Vic 3083 Australia

architectures, determined by a combination of application class and domain, and that these are used by designers as an aid in their work.

At the same time, the final "architecture" of a system is the result of beginning with some early-stage design, and following a series of steps which transform this into working code. This raises the question that this position paper addresses "given a particular a priori (design) architecture, to what extent will particular methodologies ensure that this will be evident in the finished code?". In what follows, we explore this issue briefly and suggest some criteria that may be of use in assessing this relationship. In addition, comments will be made on the results obtained by recent work in re-engineering which explores the relationship between a priori and final architecture.

2. The Problem of Architecture Preserving Properties of Development Methods

Our starting point is that, if an architecture can be determined in the early stages of a project, then one would like to see it appear in the final implementation. For this to occur, at least two conditions must be met:-

A/ The a priori architecture must be such that, if implemented, it will realise a system which conforms to the functional (and non-functional) specification,

B/ The result of implementing the system beginning with the specification and architecture, should be such that the architecture is preserved in the final code.

We can add a corollary to this:-

C/ If B/ cannot be guaranteed, then the implementation derived architectural drift[4] **[16]** should be predictable.

The first of these conditions can be considered "weak". To be meaningful, we need to restrict it to the "logical" architecture (Krutchen **[14]**), since a system may be implemented on (for example) a message-passing service, or a component communications brokering systems such as CORBA. A choice such as this highlights the multiple architecture problem described by Baragry,**[1,2,3]**, Krutchen **[14]** and others. Nevertheless, such a condition is needed if the concept of architecture is to have meaning at all, since an implemented system should have the a priori architecture. More importantly, condition A will allow some meaning to be put to situations where the desired outcome (architecturally) is not achieved - it may be that the functionality could not/should not have been implemented using the proposed architecture, and this was recognised during its development. (See **[17]** for an example where an a priori architecture was subsequently deemed to be unimplementable).

[4] We prefer the term "drift" to that of "erosion". In our contexts, we apply these terms differently to Perry and Wolf [16], and prefer to apply "erosion" to post-completion change due to maintenance or evolution, and "drift" to departure from the a priori architecture,

Our second condition in fact specifies the hypothesis that we wish to test. It considers the implementation process to be a (series) of transforms applied to some (original) specification (in this case, including an a priori architecture) which yield an executable system. This concept is not new (see for example **[24]**).

For a development process to be successful, we require that the completed system be judged to meet the requirements embodied in the specification. Again, there is nothing novel in this statement. We now extend this to what we call "the architecture preserving" property as described above. However, it is well known that systems generally are subject to significant amounts of maintenance and, we generally assume that this will be more efficient if there is documentation which describes the implemented system. Since the a priori architecture is part of this documentation, we assume that, if condition B/ held, then maintenance of the system should be easier.

Finally, our corollary C/ allows some judgement to be made at the beginning of a project as to the risk presented by the combination of early assumptions relating to architecture and development methodology. It may even permit predictions as to the implementation architecture.

3. The Issue of Design Architecture Persistence

It is apparent that we are addressing two aspects of what could be called "Architecture Persistence". Firstly, the extent to which the architecture persists during the implementation, and secondly, the way in which the architecture persists over time as a system is maintained. The second issue has been the subject of considerable study for some time and is now known as "architectural erosion"[5] (see Perry and Wolf**[16]**)), however much of this is in the context of re-engineering (see Ding and Medvidovic(**[10]**) for example).

Baragry's **[1]** case study of the implementation of the HyperEdit sub-system of the Amdahl Australian Intelligent Tool Program's HyperCASE (Cybulski and Reed **[9]**) showed that the architecture of this system evolved substantially during its development and maintenance. This supported earlier conjectures by Reed[6] **[18]**, based in his experience with the TAME project at the University of Maryland in 1986 (See Basili and Rombach**[5]**). Reed observed that, despite the efforts of skilled designers, the system architecture seemed to evolve. The primary problem seemed to be that, despite the a priori understanding of the interaction of the sub-systems in TAME, new interactions (and hence different architecture) became apparent as the design of relatively large subsystems progressed. In the case of the TAME system, each of the subsystems was extremely large and there were un-answered research questions to be solved before an implementable design could be completed (in fact, several were the subject of a PhD theses). The problems being "solved" in each of the sub-systems were largely independent of the overall architecture up to a certain point. Algorithms for metric generation and composition, had to be developed, canonical representation of control-flow graphs for programs written in different languages (which were to be capable of being processed by language independent metric tools), and a GQM (Basili **[4]**) tool had to be developed. Much of the design effort therefore was independent of the details of common services, and even their interconnection. As the designs unfolded, two contradictory trends became apparent. Firstly, a substantial amount of the design and implementation could progress without the architecture having much impact. Secondly, as sub-systems "understanding" of the nature of services they needed evolved (which it did as various problems were solved), they dictated changes to the overall architecture. This lead to the conjectures enunciated by Reed in 1987 to the effect that there were cases where an architecture either could not be determined early in a project, or where this was unnecessary.

In terms of the problem posed, the experience of systems of the TAME/ HyperEDIT class, are probably not good examples. They do, however, establish that the problem exists. They may also provide a basis for identifying classes of system where architecture preservation or persistence is either difficult or not possible. We should point out that a significant percentage of the re-engineering publications on architecture recovery make mention of this "architecture drift", but few relate it to any specific factor. In addition, we are now drawing attention to the role that development methodologies and processes may play in this.

---

[5] See footnote 2.

[6] Reed did not draw directly on the TAME experience in his 1987 address, however, it was in fact the stimulus for the conjectures.

4. Research on the Relationship Between A'priori Architecture and Implementation Architecture-Architecture Recovery and Erosion

Recently, the re-engineering and maintenance communities have made architecture recovery, and its related topic, architecture erosion, major areas of investigation. Despite the wide-ranging nature of this work, and its quality and value, we were had difficult locating material specifically addressing the problems of interest here (although we do not regard our search as exhaustive). Indeed, only a modest number of papers made much mention of a priori documentation containing an architecture description, and many of those were noting that it was absent, or did not reflect the final architecture. Ding, and Medvidovic **[10]**, for example, comment on some current approaches to development which, in their view, focus on an Object Oriented Programming Language and Interactive Development Environment, and hence may have no a priori architecture for maintainers to work from. Mendoca and Kramer **[15]** observe that the gap, in their view, between high-level abstractions and implementation is cause by the difference between the representational concepts available at both levels. Kazman and Carriere **[13]** note the hopelessness of the maintenance task if "*..the as-implemented architecture does not match the as-designed architecture..*" and go on to propose view extraction and view fusion as a possible means of maintaining appropriate documentation. Rugaber and Wills **[21]** did not include architecture preservation as in issue in their excellent paper on research infrastructure for re-engineering. Ran and Kuusela **[17]** reporting on the ARES project, reported examples of two architecture designs which were not implementable. One of these was chosen for (development) process related reasons.

Perhaps the harshest comment we could find came from Harris et. al **[11]**, "*While it is clear that every piece of software conforms to some design, it is often the case that existing documentation provides little clue to that design. For example, while the system block diagram portrays an "idealized" software architecture, it typically does not even hint at the source level building blocks required to construct the system"* (op. cit).

The need for research of this kind is supported by Bennett and Ralvich's **[6]** contribution to the futures of Software Engineering Symposium in Limerick 2000 in which they advocated an evolutionary development model, over a product's lifecycle. While not new (a far more limited approach can be found in Royce's original paper from which the waterfall model is taken (Royce 1970 **[20]**, although this deals primarily with the initial development) they point out that architectural adaptability and evolvability are as, a result major research issues. We add that the issue of preservation also needs to be considered.

We would be remiss if we did not mention work by Shaw et al **[22]** which describes graphically supported architecture description language capable of automatically generating executable systems given appropriately defined components and connectors. A tool such as this is inherently architecture preserving. In addition, there is work in the re-engineering community which explores and attempts to explain the differences between a priori and implementation architectures. See for example, the paper by Hassan and Holt **[12]** which is an exception in that it describes the

reference to conceptual architecture mappings for three web servers. Their conceptual architectures are developed from a combination of code and documentation, but it is not clear how these relate to any original architecture. They do, however, report explanations for the lack of conformance between the reference-conceptual architecture pairs.

5. Conclusions and Acknowledgements

We have put forward a prima' face' case for extending current architecture and process-related research. In doing so, it is implicit that some guidance would be forthcoming for inclusion in any guidelines for software architecture assessment. This is particularly important if Bennett and Ralvich **[6]** are in fact correct. Equally importantly, we argue that investigations of this kind will assist in addressing a wider range of software engineering problems, including the place of software architecture, and, the nature of software development processes.

The authors wish to acknowledge discussions with Daniela Mehandjiska-Stavreva and Phil Stocks of the School on Information Technology and Bond University, and the first author gratefully acknowledges the support for the visiting position which is making this and other work possible. Ultimately, any errors etc, are the responsibility of the authors.

6. References

[1] Baragry, J(2000) Understanding Software Engineering: From Analogies With Other Disciplines To Philosophical Foundations. PhD in Computer Science and Computer Engineering La trobe University (Submitted 2000) Chap 4. AN EXAMPLE OF UNDERSTANDING BASED ON THE ARTEFACT ENGINEERING VIEW – SOFTWARE ARCHITECTURE

[2] Baragry, J. and K. Reed (1998). Why Is It So Hard To Define Software Architecture? Proceedings Asia Pacific Software Engineering Conference, Tapei, Taiwan

[3] Baragry,J and Reed, K(2001) why we need a different view of software architecture WICSA 2001

[4] Basili, V. R.,(1985) Quantitative Evaluation of Software Methodology, Keynote Address, First Pan Pacific Computer Conference, Melbourne, Australia, September, 1985.

[5] Basili and Rombach, H.D.(1988), The TAME Project: Towards Improvement-Oriented Software Environments, IEEE Transactions on Software Eng., vol. 14, No. 6, June 1988.

[6] Bennett, K.H. and Rajlich, V.T. Software Maintenance and Evolution: a Roadmap Future of Software Engineering Symposium Limerick Ireland 2000 publ ACM pp73-87

[7] Brooks, F. P. (1962). Architectural Philosophy. Planning a Computer System - Project Stretch. W. Buchholz (eds), McGraw-Hill**:** pp. 5-16.

[8] Brooks, F. P. (1975). The Mythical Man-Month: Essays in Software Engineering, Addison-Wesley Publishing, 1975

[9] Cybulski , J.L.C. and Reed,K (1992) A HYPER-TEXT BASED SOFTWARE ENGINEERING ENVIRONMENT IEEE Software, Vol. 9 No. 2, Mar 1992 pp 62-68

[10] Ding,L. and Medvidovic, N (2001) Focus: A Light-Weight, Incremental Approach to Software Architecture Recovery and Evolution Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA'01)

[11] Harris, D.R., Reubenstein, H. B. and Yeh, A.S .Reverse Engineering to the Architectural Level ICSE '95, Seattle 1995, ACM pp186-195

[12] Hassan, A.E. and Holt, R.C A Reference Architecture for Web Servers Proceedings of the Seventh Working Conference on Reverse Engineering (WCRE'00), Brisbane 20002, IEEE Computer Society

[13] Kazman, R. and Carriere, S.J(1998). View Extraction and View Fusion in Architectural Understanding Proceedings of the Fifth International Conference on Software Reuse, Victoria, 1998, IEEE Press

[14] Kruchten, P. (1995). Architectural Blueprints - The "4+1" View Model of Software Architecture. IEEE Software(November 1995).

[15] Mendoca, N. C. and Kramer, J(1996) . Requirements for an Effcetive Architecture Recovery Framework Joint proceedings of the second international software architecture workshop (ISAW-2) and international workshop on multiple perspectives in software development (Viewpoints '96) on SIGSOFT '96 workshops October 1996, ACM 1996

[16] Perry, D.E and Wolf, A.L (1992) Foundations for the Software Architecture ACM SIGSOFT Software Engineering Notes Vol. 17 No. 4 Oct 1992 pp. 40-52

[17] Ran,A and Kuusela,J (1996) Selected issues in architecture of software intensive products Joint proceedings of the second international software architecture workshop (ISAW-2) and international workshop on multiple perspectives in software development (Viewpoints '96) on SIGSOFT '96 workshops October 1996

[18] Reed, K. (1987). Commercial Software Engineering, The Way Forward. (keynote address). Keynote address Australian Software Engineering Conference, Canberra. ACT. Australia. 1987

[19] Reed,K (2000) THE FUTURE OF SOFTWARE ENGINEERING AND RE-ENGINEERING AS SOFTWARE ARCHEOLOGY. Keynote Speech to the 2000 Working Conference on Software Engineering, Brsibane, Nov. 2000

[20] Royce, Winston W(1970), Managing the Development of Large Software Systems IEEE WESCOMN 1970 p-19

[21] Rugaber, S and Wills L. M. Creating a Research Infrastructure for Reengineering 3rd Working Conference on Reverse Engineering (WCRE '96) Monterey November 1996 IEEE Computer Society

[22] Shaw, M., DeLine, R., Klein, D. V., Ross, T. L., Young, D. M. and Zelesnik, G.(1995) Abstractions for Software Architecture and Tools to Support Them IEEE Transactions on Software Engineering Vol. 21 No. 4 Apr. 1995 pp 314-335)

[23] Shaw, M. Large Scale Systems Require Higher-level Abstractions Proceedings of the Fifth International Workshop on Software Specification and Design, IEEE Computer Society, 1989 pp 143-146

[24] Wileden, J. C. (1986). This is *IT*: A Meta-Model of the Software Process. *ACM Software Engineering Notes*, 11(4), 9-11 Proceedings of the International Workshop on the Software Process and Software Environments, Trabuco Canyon March 1986