Software Archaeology - An interdisciplinary view

Gerhard Chroust
Systems Engineering and Automation
Kepler University Linz
Systems Engineering and Automation
gc@sea.uni-linz.ac.at

Abstract

Maintenance is one of the key problems of software engineering. Because of analogies to archaeology software maintenance is often called 'software archaeology'. This paper discusses analogies between software maintenance and archaeology, emphasising similarities and dissimilarities. It shows some surprising parallels and insights concerning what one calls legacy systems and archaeological artefacts, respectively. The paper also indicates where these two - so vastly different areas can learn from one another.

The Legacy Problem, Maintenance and Knowledge Elicitation

Maintenance of software products is a key problem today, i.e. repairing and enhancing so-called 'legacy systems'. Industry is spending more than half of a software products life time costs on maintenance (End, Gotthardt and Winkelmann 1986). Legacy systems have outlived their planned useful life, their programmers, their base technology etc. The general public recognized this in the course of the transition to the year 2000 and (for Europe) during the transition to the Euro.

There are several causes why software becomes less usable or erroneous during its life time (Basili 1990, Lehmann and Belady 1985).

These causes should sound familiar also to archaeologists:

- (Lehman's Law) Successful systems have to change in order to remain acceptable to their users (Belady 1985).
- Stable systems which do not need change are 'dead' systems.

- Most of the changes are not caused by programming errors, but are due to external changes (changed legislation, different requirements, unforeseen changes in the environment, e.g. changing to the Euro, changing to a different operating system, etc.). Industry sources indicate that approx. 40 % of all changes (i.e. 'maintenance') are actually changes to adapt a system to changed environments and external requests (Sneed 1990).

One has also to admit that legacy systems have several advantages which distinguish them from systems to be newly written and therefore justify maintenance:

- Existing and operational systems often contain considerable hidden domain knowledge not documented or know to the users.
- Old system work, which is not self-evident for newly built systems.
- Users are used to the old system and know to how to handle it, and even how to avoid/circumvent certain problems and errors.
- etc.

Therefore legacy systems are not only old burdens, but also old treasures - like in archaeology (Fig. 1)

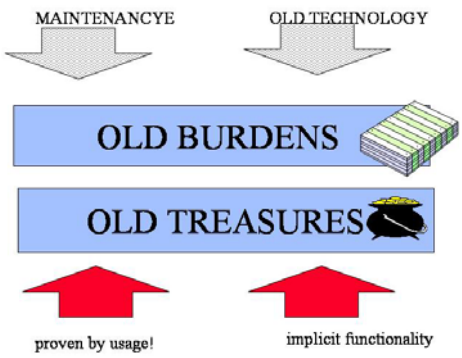

Fig. 1: Old Burdens and Old Treasures

They pose, however, some problems:

- Their developers don't exist any more, one cannot ask them questions about the system, the motives etc.
- Documentation (for design and operation) is non-existent, is lost, is unreadable, or written in nowadays unknown language (Who still can read programs written in the programming language IPL-V? Or who can read Norwegian runes?).
- Existing documentation is unreliable and often outdated with respect the current system in operation.

- The requirements, motives, objectives and the environment under which the systems were build and operated do not exist any more or cannot be understood.
- The use or purpose of the system is misunderstood due to preconceived opinions of the maintenance engineer: A typical book about archaeology carries the (translated) title "They found what they knew - Archaeology as a mirror of modern times" (Rehork 1989) and Arthur Evans' interpretation of Knossos is also the target of heavy criticism.
- Parts of the system are missing and forgotten or have been reconstructed.
- These systems contain extra parts which are not useable any more, even not accessible in normal use (software calls this 'dead code'), but still making understanding more difficult.
- Large parts of the system have been changed over and over again in the course of their use.
- The original (probably clear design) has been modified over time and was obliterated by various minor modifications.
- The systems were are build in a technology with is outdated and often not safe and reliable any more.

We recognize that one of the major problems is acquiring enough knowledge about the legacy system based on the evidence of available artefacts. In a publication on Software Architecture Recovery we read (Philippow, Pashov and Riebisch 2003): *The available evidence in a legacy [software] system often is not sufficient for its understanding and recovery. In most cases the [software] documentation is outdated and poor. It is possible to argue that the most reliable information is in the [source code / bricks]. Nevertheless a significant knowledge about the problem domain is required to improve the facility for extraction of useful architectural information.* Delete twice the word *software* and replace the word *source code* by *original bricks* and it could have been written by some archaeologist.

One has to *elicit knowledge* from the available sources, *structure it* and *preserve it* in adequate and hopefully better accessible and understandable form. Archaeologists fight with the same problems (Hunt and Thomas 2002), therefore Harry Sneed, a well known German-Canadian software pioneer, coined the term 'software archaeology' (Sneed 1994) for maintenance work in the software industry (Hunt and Thomas 2002, Dennett 1986). We will develop this idea further.

A major distinguishing characteristic is the *aim* of these two fields:

- Archaeology puts the emphasis on *putting the observer into the historical, "original" environment'*, striving to preserve the past for analysis and contemplation, while
- software maintenance tries *'to bring the legacy system into today's users' environments'*, striving to keep old systems in productive use.

Handling Legacy Systems - Re-techniques

Software engineering provides a range of techniques to handle legacy systems, the so-called *'re-techniques'* because of their common prefix *(in italics you find remarks targeting archaeology)*.

REcognition: Recognizing and identifying useful information (using data mining and pattern recognition) are important methods in software maintenance. *This turns out to be more difficult in archaeology because most of the data are hidden and difficult to locate*. Software engineers, however, often are also at a loss to find a certain critical software module's imbedded source code, taken from a perhaps obsolete library.

REallocation: Artifacts are often brought into another environment. In software engineering the reason is mostly change of computer hardware, or operating system etc. *Reallocation in archaeology was often for safeguarding or protection from destruction, be it from environmental dangers, be it from robbers, etc. Often the reallocation itself was done for pure greed. Especially in archaeology reallocation is difficult, cumbersome, error-prone and not always successful: artifacts get lost, broken, stolen, and confused during transfer*

REcombination: Related parts are not necessarily in one place, legacy software has functions distributed over the code, largely due to maintenance patchwork. *Finding the head fitting to a headless statue needs considerable human intuition, but can be supported by massive use of computers, typically to identify potentially matching pieces and interfaces dispersed over the world.*

REpair: Artifacts need repair in order to preserve them, a fact well-known in software engineering, where maintenance is important but also difficult due to software's idiosyncratic properties like invisibility and ease of changing (Brooks 1986). When repairing software the original code has only to be preserved if there are old systems still using it and this can easily achieved by copying. *In archaeology preservation of original artifacts and the precise distinction between original and replacement is a key concern, especially in the case where a site has many strata*.

Restoration: *This is one of the major challenges and source of controversy in archaeology (but not an issue in software): to which epoch and status should the artifact be restored? Typically after the fire in the famous Redoutens„le of the Vienna Hofburg, the discussion arose whether to restore these rooms to their last 20th century appearance or to their original appearance (1705)*. Software can easily be duplicated to allow both versions to exist in parallel. *For archaeology only Virtual Reality* (Billinghurst and Kato 2002) *offers the chance to view several views*.

RE-documentation: Traditionally (not only in archaeology) documentation of artifacts is rudimentary, often not existing and unreliable or unreadable. Some documentation is not even recognized as such: initially even cuneiform inscriptions were misunderstood as decorations without deeper semantics. *Fortunately archaeologists are trained in documentation and see this as one of their major professional tasks - in contrast to the archaeological adventurers of the 18th century and in contrast to most software engineers*.

RE-structuring: Due to maintenance the structure of a software product is gradually deteriorating because changes or often done without concern (or knowledge) of the initial grand plan (architecture). *In archaeology this is often the result of changed usage (e.g. churches, especially belonging to other religious, are used as stables or for other profane purposes*.

- It is therefore necessary to re-structure a software product, compatible with the original concepts and the changes made since.
- The structure and organisation might - sometimes even unintentionally - be completely transformed into another structure. *This effect is well known to archaeologists, when different uses of a buildings cause more or less small changes which in sum, however, often completely change*

the outlay, the appearance and the usage pattern of a building. It may go so far as to use the Acropolis as an ammunition depot with the effect of being blown up during one of the frequent battles.

REverse Engineering: Key issues when confronted with some unknown artefact are: What does it accomplish? How does it function? What was its purpose? Why has it been built like that? *Archaeologists, perhaps more than software engineers, understand that these questions have to be answered on different semantic levels: If we know what the form of a house was (of which we may not see more than the foundations), we still do not know what rites or professions were performed in the various rooms, let alone why a certain ceremony was performed at all. And we know that more than one interpretation is possible.*

REengineering: Reengineering is the rebuilding of a system with different means and/or technology carrying over the information or functionality of the old system but for sustained/improved usage. *Archaeology has the privilege not to have to cater for current usage of most archaeological artefact. Such use would be contra-productive and destructive for scientific research of archaeological sites*.

Nevertheless in some rare instances even archaeology has to use reengineered artefacts. Examples are copies of statues from medieval churches or the duplication of the caves of Lascaux in Paris.

A true synergy between archaeology and computer technology is Virtual Reality (Billinghurst and Kato 2002, Forte and Siliotti 1997, Stone 1992) which allows us to truly re-engineer historical artefacts: instead of stone and masonry, they are rebuild in bit and bytes. If done with perfection (at a cost not to be underestimated!) we can have the same 'look and feel' of the 'virtually re-build' artefact. sometimes even indiscernible from the original artefact. This allows many people to see and to even touch(!) it, often without leaving their home and allows several versions to exist in parallel.

REuse in different context: In software production, as in every other industry, the reuse of partial products is one of the keys to productivity and quality (Allen 2001, Brown 1996). *With respect to archaeology this use is a highly undesirable human activity since it usually means carrying away*

archaeological artefacts for some other unknown, profane use, like using the Pyramids or the Roman castellum at Carnuntum as cheap source of building material like a stone quarry.

We can compare archaeology and software development with respect to the need to apply the re-techniques (Fig. 2). We have used '++' and '+' to indicate strong and weak usage.

technique	archaeology	software development
REcognition	++	
REallocation	+	++
REcombination	++	
REpair	++	++
Restoration	++	+
RE-documentation	++	++
RE-structuring		+
REverse Engineering	++	+
REengineering		++
REuse in different context		++

Fig. 2: Different Re-techniques and their relevance for achyeology and software development

Cross-Fertilization

We can observe that the maintenance of software products and archaeological work have considerable overlap, especially with respect to the following problems (Hunt and Thomas 2002):

Preservation Problem: The identification and preservation of artefacts, is of utmost importance. In archaeology - due to the uniqueness of artefacts - it has to be done with utmost precaution. In software engineering the production of identical copies or several versions is no problem, and is practised in industry, were many versions of the same product are in use at the same time.

Understanding Problem: The understanding of the meaning of the available documentation and its validation is a key challenge, misinterpretations often

are long-living (cf. A. Evans' interpretation of Knossos and the purpose of different rooms) and counter-productive.

Documentation Problem: Reading and understanding ancient documentation is a multi-levelled problem, starting from isolating characters and words (Doblhofer 1990) to trying to read and pronounce correctly the utterances etc. Mistakes are to be expected, like queen "Schu-ad" (Woolley 1929), which is now read as "Puabi" (Strommenger 1962). Programming languages do not pose deciphering problems like some of the ancient languages. In the domain of software lack of proper documentation, lack of visibility of the dynamics of a program causes problems by forcing maintenance engineers to deduct bottom-up the functionality of a program.

Matching Problem: In any complex system a key to understanding is knowing the relation of artefacts to one another. *Archaeology has the disadvantage that many of the artefacts have been removed from their original site (often illegally and secretly), and have gone through many hands (and countries!). Establishing the original relationships needs modern technology and algorithmic approaches to pattern matching and data mining, only possible nowadays.*

Completion Problem : Archaeology is hampered be the problem of missing parts. One of the 'noblest' ways to humiliative a beaten enemy was to disgrace his effigies, dismember his statues etc.. As Afghanistan has shown, this is not a prerogative of a dark past of humanity. The question is how to complete the 'picture' at least as far as the understanding of the while is necessary, even if not every detail is restored. Who is the other person on the broken slab of stone? etc. In software we sometimes run in the same problem: when moving a program to a different computer some auxiliary programs become inaccessible.

Reverse Engineering Problem: As explained in section 2 the recreation of the original design and architecture is difficult and - what is even worse - ambiguous. Actually we are forced to make some hypotheses, which in themselves (due to the Understanding Problem) are very likely biased by our preconceptions.

Configuration Problem: An archaeological site usually consists of several layers with different contents. But changes in nature often disturb the sequence of strata. Even finding an artefact in a certain stratum does not guarantee its synchronicity, it could be much older and only been abandoned much later.

Presentation Problem: For different reasons both archaeology and software have a similar problem: How to explain to outsiders (including those who can provide the necessary sponsoring) what actually the underlying structure, concepts, and plans were. For software this is mainly caused by its invisibility (Brooks 1986) and the difficulty to show dynamic behaviour. *For archaeology the problem is often the lack of some important parts of an artefacts (very often the head!). Virtual Reality or Mixed Reality can be very supportive in virtually present a complete image* (Billinghurst and Kato 2002, Stone 1992, Forte and Siliotti 1997, Tarumi, Morishita, Ito and Kambayashi 2000).

In Fig. 3 we indicate the direction of influence between archaeology and software development. The influence can be of technological nature (software) or of intuitive, conceptual nature (largely archaeology).

problem	archaeology	influence	software engineering
Preservation Problem	Physical preservation	none	logical preservation
Understanding Problem	visible evolution of churches, palaces etc.	intuitive understanding →	invisible evolution of structure of programs
Documentation Problem	nonexistence, incompleteness	← administrative help	nonexistence, unreliability
matching problem	manual, optical means	← technological support	search algorithms, scanning, internet
Completion Problem	cultural understanding problem	none	logical problem
Reverse Engineering Problem	only single version feasible	← representational help intuitive understanding →	virtual reality , multiple versions possible
Configuration Problem	changing use of buildings	intuitive understanding →	changing composition of product
Representation Problem	preserving historical versions, indicating missing parts	← technological support	virtual reality/mixed reality

Fig. 3: Archaeology and Software Development - Cross-Fertilization

Summary

In this paper we have shown some similarities between the field of Software Maintenance, often called *"Software Archaeology"* and Archaeology. We see the chance for a synergy between the two fields:

- Archaeology can be helpful in providing understandable, obvious examples for the often abstract, ephemeral observations and problems of software (due to software's basic invisibility). Some of the problems where software engineers have difficulties of accepting them intuitively are obvious in archaeology, e.g. the destruction of structure by maintenance, the drifting of architecture by enhancements, the ambiguity of reverse engineering, etc.).
- Software can offer new methods and technology into the long-established field of archaeology making some tasks feasible which were outside the reach of purely manual approaches, like finding the matching head to a headless figure by comparing every one with every other. Archaeology can profit from software's ability to process masses of data and supplying new representational means by Virtual and Mixed Realities.

Like in many other fields, interdisciplinarity pays off.

References

ALLEN, P., 2001. Realizing e-Business with Components. Addison-Weseley, ISBN 0-201-67520-X.

BASILI, V.R., 1990. Viewing Maintenance as Re-use oriented Software Development. In *IEEE Software* Jan. 1990:19-25.

BELADY, M.M., 1985. Program Evolution: Processes of Software Change. In *London Academic Press*.

BILLINGHURST, M. and KATO, H., 2002. How the virtual inspires the real - Collaborative augmented reality. In *CACM* Vol. 45, No. 7:64-70.

BROOKS, F.P.Jr., 1986. No Silver Bullet - Essence and Accidents of Software Engineering. In: Kugler, H.J. (ed.), *Information Processing* 86*, IFIP Congress*:1069-1076.

BROWN, A.W. (ed.), 1996. Component-Based Software Engineering. IEEE Computer Society, ISBN 0-8186-7718-X.

DENNETT, D., 1986. Julian Jaynes' Software Archaeology. In *Canadian Psychology,* Vol. *27*, April (2):149-154.

DOBLHOFER, E., 1990. Zeichen und Wunder. Weltbild-Verlag, ISBN 3-89350-X.

END, W., GOTTHARDT, H. and WINKELMANN, R., 1986. Softwareentwicklung - Leitfaden für Planung, Realisierung und Einfuehrung von DV-Verfahren. 5th ed., Siemens AG.

FORTE, M. and SILIOTTI, A., 1997. Die neue Archäologie - Virtuelle Reisen in die Vergangenheit. Gustav Lübbe, ISBN 3-7857-0888-2.

HUNT, A. and THOMAS, D., 2002. Software Archaeology. In *IEEE Software* Vol. 19, No. *2*:20-22.

LEHMAN, M.M. and BELADY, L. (ed.), 1985. Program Evolution - Processes of Software Change. *APIC Studies in Data Proc.* No. 27, Academic Press.

PHILIPPOW, I., PASHOV, I. and RIEBISCH, M., 2003. Application of Feature Modeling for Architecture Recovery. In *Softwaretechnik - Trends* Vol. 23, No. *2,* ISSN 0720–8928:19-20.

REHORK, J., 1989. Sie fanden, was sie kannten - Archäologie als Spiegel der Neuzeit. Bastei Lübbe, ISBN 3 404 64082 9.

SNEED, H., 1994. Claimed to have coined the term 'Software Archyeology - Personal Communication.

SNEED, H.M., 1990. Software Wiederverwendung. In ADV (ed.), *EDV in den 90er Jahren: Jahrzehnt der Anwender - Jahrzehnt der Integration ADV:*199-214.

STONE, R., 1992. Virtual Reality and Telepresence. In *Robotica* No. 10:461-467.

STROMMENGER, E., 1962. Fünf Jahrtausende Mesopotamien. Hirmer München.

TARUMI, H., MORISHITA, K., ITO, Y. and KAMBAYASHI, Y., 2000. Communication through virtual active objects overlaid onto the real world. In *Proceedings of the third international conference on Collaborative virtual environments*, ACM Press, ISBN 1–58113–303–0:155-164.

WOOLLEY, L., 1929. Vor 5000 Jahren - Die Ausgrabungen von Ur und die Geschichte der Sumerer. 2nd ed., Franckh'sche Verlagshandlung, Stuttgart.