Software Archaeology - An Interdisciplinary View

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

See the CD for the extended version

Abstract

Maintenance is one of the key problems of software engineering, often nicknamed '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 or legacy artefacts.

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'. These 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) in the change to the Euro.

There are several causes why software becomes less usable or erroneous during its life time (Lehman 1985, Basili 1990). These causes should sound familiar also to archaeologists:

- (Lehman's Law) Successful systems have to change in order to remain acceptable to their users (Lehman 85).
- 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 change to the environment, e.g. changing to the Euro). Industry sources indicate that approx. 40% of all changes (i.e. 'maintenance') are actually changes to adapt a system to changing environments (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.
- etc.

Therefore legacy systems are not only old burdens, but also old treasures - like in archaeology. They pose, however, some problems:

- Their developers do not exist any more, one cannot ask them.
- Documentation (for design and operation) is non-existent, is lost, is unreadable, written in nowadays unknown language (Who still knows the programming language IPL-V?).
- Existing documentation is unreliable and often outdated with respect to the current system in operation.
- The requirements, motives, objectives and the environment in which the systems were build and operated do not exist any more or cannot be understood.
- Parts of the system are missing and forgotten.
- These systems contain extra parts which are not useable any more, even not accessible by normal operation (software calls this 'dead code'), but make understanding more difficult.
- Large parts of the system have been changed over and over again.
- The systems were are build in a technology with is outdated and often not safe and reliable any more.
- The original (probably clear design) has been modified over time and was obliterated by various minor modifications.

We recognize that one of the major problems is acquiring enough knowledge about the legacy system using available artefacts. 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 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 2002, Dennet 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 users' environments', striving to keep old systems in productive use.

Handling Legacy Systems

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 targeted at archaeology).

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

REallocation: Artefacts often are brought into another environment, mostly for safeguarding or protection from destruction, in archaeology often out of pure greed. *Especially in archaeology this is difficult, cumbersome, error-prone and not always successful: artefacts get lost, broken, stolen and confused during transfer.*

REcombination: Related information is not necessarily in one place, legacy software has functions distributed over the code, largely due to maintenance patchwork. *Human intuition can be augmented by massive computer support to identify potentially matching pieces and interfaces dispersed over the world.*

REpair: Artefacts 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 easy changeability (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 artefacts 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 artefact be restored? Typically after the fire in the famous Redoutensäle 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 2002) offers the chance to see several views.*

RE-documentation: Traditionally (not only in archaeology) documentation of artefacts 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.

- 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.*

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 discern only the foundations), we still do not know what rites or professions were performed in the various rooms, let alone why a certain activity 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 blend between archaeology and computer technology is Virtual Reality, which allows us - in a true re-engineering fashion (same 'look and feel' but completely different technology) to 'virtually re-build' every artefact sometimes even indiscernible from the original artefact in order to enable many people to see and to even touch (!) *without leaving their home.*

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. *With respect to archaeology this is a highly undesirable human activity since it usually meant carrying away archaeological artefacts for some other unknown, profane use, like using the Pyramids or the Carnuntum site as a cheap source of building material like a stone quarry.*

Cross-Fertilization

We can observe that the maintenance of software products and archaeological work have considerable overlap, especially in the following areas (Hunt 2002):

Preservation Problem: The identification and preservation of artefacts, etc. is of utmost importance. In archaeology - due to the uniqueness of artefacts - it has to be done with utmost precaution, the production of identical copies eases this problem for software engineering.

Understanding Problem: The understanding of the meaning of the available documentation and its validation is a key challenge, misinterpretations often are long-living (cf. interpretations of Knossos) and counter-productive.

Documentation Problem: Reading and understanding ancient documentation is a multi-levelled problem, starting from identifying characters or symbols (Doblhofer 1990) to trying to read and pronounce the utterances etc. Programming languages do not pose similar 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.*

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 sponsor money) what actually the underlying structure, concepts, and plans were. For software this is mainly caused by the invisibility of software and the difficulty to show dynamic *behaviour. For archaeology part of the problem is the lack of some important parts of an artefacts. Virtual Reality or Mixed Reality can be very supportive there (Billinghurst 1992, Forte 1997, Tarumi 2000 and Stone 1992).*

Summary

Archaeology can be helpful in providing understandable, obvious examples for the rather abstract, ephemeral observations and problems of software, while software can bring new ideas and technology to the long-established field of archaeology, introducing new approaches and methods. In this paper we have shown some similarities between the field of Software Maintenance (*"Software Archaeology"*) and Archaeology. Some of the problems where software engineers have difficulties to accept them on an intuitive basis, are obvious in archaeology (e.g. the destruction of structure by maintenance, the drifting of architecture by enhancements, the ambiguity of reverse engineering, etc.) On the other hand 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

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

BILLINGHURST, M. and KATO, H., 2002. How the virtual inspires the real - Collaborative augmented reality. 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.

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

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

FORTE, M. and SILIOTTI, A., 1997. Die neue Archäologie - Virtuelle Reisen in die Vergangenheit. Gustav Lübbe (ISBN 3-7857, 200888-2).

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

LEHMAN, M.M. and BELADY, L.A., 1985. Program Evolution - Processes of Software Change. APIC Studies in Data Proc. no. 27, Academic Press.

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

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

STONE, R. Virtual Reality and Telepresence. Robotica no. 10:461-467.

TARUMI, H., MORISHITA, K., ITO, Y. and KAMBAYASHI, Y. Communication through virtual active objects overlaid onto the real world. Proceedings of the third international conference on Collaborative virtual environments, ACM Press:155-164.