Global perspectives on legacy systems

Magnus Ramage[1]
Centre for Complexity and Change,
Faculty of Technology,
Open University
m.ramage@open.ac.uk

This paper describes ideas which arose at two expanded workshops of the SEBPC legacy systems theme. In an earlier paper (Ramage and Munro, 2000) we discussed the early results of that theme; to a large extent, this work follows on from that paper and should be read in conjunction with it.

The two workshops discussed here were held roughly a year apart, at international conferences meeting in the UK. The first workshop (the seventh in the SEBPC legacy series) was held on 30 August 1999 in Oxford, at the International Conference on Software Maintenance (ICSM), which attracts a mostly technical participation, from software engineering academics and from industrial practitioners and consultants. The second workshop (the eleventh legacy workshop) was held on 4 July 2000 in Stafford, at the International Conference on Enterprise Information Systems, a more mixed collection of computing and business academics (but no industrialists). Roughly 15 people attended the first workshop, from 3 countries - 13 academics and 2 industrialists; 9 of these were connected with SEBPC, 6 were not. In the case of the second workshop, 12 attended in total, from 5 countries (all academics); 5 were connected with SEBPC, 7 were not.

In the earlier paper, we discussed the nature of legacy systems, characterising them as follows:

- It is important to distinguish between legacy software and legacy systems (a wider concept, which includes the organisation within which the software is situated, as well as its processes and members).
- 'Legacyness' lies in the gap between the needs of the business and the capabilities of the technology.
- Legacy is what is left after a particular event happens: the recognition of that event varies from place to place and from person to person.
- Legacy systems contain much which is valuable (especially data): in fact, the term only makes sense in referring to systems which are important but hard to change.
- It is as important to anticipate, and prevent, future legacy problems, as it is to correct existing ones.

It is not my intention here to reproduce that discussion, but rather to show how it has been extended in these later workshops.

The rest of this paper of this paper is divided into four sections, covering the main issues discussed at the workshops: the nature and dynamics of legacy systems, the co-evolution of software and organisations, issues around software as a technology (its engineering and its management), and organisational/people issues.

The nature and dynamics of legacy

I have described above the perspectives on the nature of legacy systems reached at earlier workshops. We heard many vivid stories from participants at the Stafford workshop about their experiences with legacy systems - of very big systems that only held together because people were continually patching them up or being employed to deal with exceptions that the system couldn't handle; of old technology that (15 years later) still worked fine for its originally-intended purpose, but was hopeless when one tried to communicate with others; of technologies regarded as ancient and useless in one place but state-of-the-art in another.

---

[1] When the workshops described here were conducted, the author was at the Research Institute in Software Evolution, University of Durham.

Is legacy just about fads and fashions? The populist view of legacy - seen in glossy computer magazines and attributable to the industry's need to sell more products - keeps arising. In some places (especially the USA) unless one is seen to have the latest hardware and software, one is behind the times and not seen as technically astute. This is not a useful view of legacy, yet it is a common one.

Two views were expressed as to a more useful perspective. Under one view, what constitutes a legacy system is entirely down to perception, that software is not legacy until someone perceives it thus. Legacy, under this view, is in the eye of the beholder. This was unacceptable to some participants in Stafford, who wanted there to be a more objective view as to what was a legacy system, and indeed regarded it as part of the task of the academic community to produce such a view. Some systems, said one person, are no longer maintainable in an economic way - they cost too much, the skill base has gone, it is too risky. To put this another way, as one participant said in Oxford, a system becomes legacy when its underlying business process has changed. It is surely appropriate in any circumstances to regard these kinds of systems as legacy.

A compromise of sorts was reached between the two views when someone else observed that such objective legacy was a property of *systems* - software in use in a given organisational/economic/people context, and not of software on its own. All agreed that it is a considerable scandal that computer scientists have yet to find good cost/time estimates for when systems become legacy in this way, or in general when it is appropriate for organisations to get rid of an old system and start again, and when they should continue to maintain that system. Recent work by Bennett and Rajlich (2000) on the dynamics of software evolution might help in this regard.

The question of the dynamics of legacy systems - the way in which they become and remain legacy - is something which has been perhaps explored less than might be appropriate. It is clear that the problems of legacy systems are “inherent, inescapable and here to stay”[2] - they are not are function of age or the technologies used, and that “today’s systems are tomorrow’s legacy”. It is also clear that “no one write a legacy system; rather they write a nice and pretty system, which latterly becomes old and ugly” (Anthony Lauder, at the Oxford workshop).

But how does that process (which Lauder and Kent (2000) have called petrification) take place? This is a subject which seems to need greater study. It is clear that it is a consequence of external change, and occurs when the wider system of which the software system is one part changes, but the software is unable to change in turn. But the dynamics require greater examination - this is a subject on which Lehman, Ramil and Wernick (1999) have also written extensively. They argue that one must look at the processes of petrification as “multi-tool, multi-level, multi-agent feedback systems”. Given the systemic nature of the issues involved in the evolution of software, we will now move on to look at the ways in which the software and organisation evolve together.

Organisations and software in mutual development

That technological change and organisational change are inexorably linked is not a new finding: it has been the main theme of the Socio-Technical Systems approach, which began in the 1950s with the redesign of coal-mines in a way that reflected new technology (Trist, 1992). That approach was led by behavioural scientists (working in what today would be called industrial psychology), looking towards technology; since then it has often been used as much by engineers and computer scientists wanting to incorporate organisational issues, especially those around ‘industrial democracy’, into the technological design process (e.g. Greenbaum & Kyng, 1991).

The socio-technical approach has its roots in the varieties of systems thinking pioneered in the 1940s. A more recent form of systems thinking has been complexity theory, especially the work of Santa Fé Institute, which looks at the complex behaviour of natural and human systems on “the edge of chaos” (Kauffman, 1995) - the ever-changing zone between stagnation and explosion. Part of the

[2] Unattributed quotations are from participants at the Oxford workshop, in an exercise where they were asked to write a phrase describing each of the five issues they regarded as most important regarding legacy systems. The collected phrases form a key part of the discussion of this chapter.

understanding presented by complexity theory is the extensive use of the concept of co-evolving systems: originally a biological model, it refers to the way an organism and its environment (or, by extension, a set of entities which form environments for each other) evolve in relation to one another. In information systems, it is clear that a software system and the wider organisational system (in a sense, its environment) change over time in relation to one another; and it is this use of co-evolution which has been important during these workshops (see Mitleton-Kelly and Papaefthimiou, 2000).

This issue recurred often at the Oxford workshop, not least because many participants were also attending another SEBPC workshop the next day on that topic (Edwards et al., 1999). One way of expressing the problematic was put by computer scientists who perceive that there is a need for a wider perspective than that found in their field, expressing this with phrases like “computer science doesn’t help with the hardest problems” and “traditional software metrics consider only the technical process and software - we need wider metrics”. Legacy, under this view, “arises from socio-technical interactions: business, organisational, technical”.

A further perspective, however, is to consider the interdependence of software and organisations (and, by extension, society). One participant said that “all software is increasingly interdependent, and human society is dependent on it” and that this “creates a moral imperative for computer scientists”. This view was especially informed at the time by the looming risks of the Y2K problem, and our collective awareness that the failure of a few crucial computer systems could have rippled through the whole society to create a massive economic and/or social collapse. Fortunately, this did not happen, but the risk of its occurrence shows the interdependence described above.

A final point about co-evolution is that, while it is mostly a self-starting process (an emergent property of the system), it depends substantially on the presence of an “enabling infrastructure”. That is, organisations and software have the ability effectively to co-evolve, but only if they are allowed and encouraged to do so by the organisational structures, processes, and culture which is in place. The devising of such infrastructure is not a trivial matter, and it remains to be understood properly (Stacey, 1996, discusses comparable infrastructures to enable creative to arise as an emergent property of organisations); but its building would go a long way towards prevention of future legacy systems.

Software engineering and software management

Of course, we must also work on specific issues of legacy systems, both technical and organisational. In this section I shall discuss what the two workshops had to say about software issues, while organisational issues are in the following section.

It is convenient from the nature of the discussions to separate software issues into those concerning the engineering of software (that is, solely technical issues) from those concerning its management and processes.

Various technical 'solutions' to legacy problems were discussed, either favourably or unfavourably. Most people at both workshops agreed that classical software re-engineering methods were no longer suitable. Techniques and tools which were suggested as alternative approaches included:

- A new class of middleware which aims to circumvent legacy problems by encapsulating old software with new interfaces that allow new capabilities to be added invisibly.
- Components solutions linked by two emerging, and competing, standards - Enterprise Java Beans and COM.
- XML, which is rapidly turning into a standard for communication, in the description of data. But it works at quite a low level, and we still need properly to understand the business processes that exchange data, and the kind of data they use.
- Determination of the nature and structure of anti-pattern and positive-pattern catalogues.
- Historical software measures: trend analysis can unveil uncontrolled and unexpected paths in the evolution of the system, and provide a support for monitoring the evolution of software attributes.
- Recovery of requirements.
- Summarisation & document archaeology agents.

- Extracting constraints from legacy information systems (these express the valid states of data objects, ensuring that only actions that result in such states may be performed).
- Techniques from database reverse engineering (which work with data and schemata).
- Since the examination of legacy systems often produces a huge amount of data, visual techniques such as three dimensional graphs and colours can help in its presentation.

Many of the different SEBPC projects aim to provide solutions to legacy software problems. We heard at some length at the Stafford workshop about the AMBOLS project (Analysing and Modelling the Behaviour Of Legacy Systems) of Staffordshire University (Alderson and Shah, 1999). The project uses techniques from organisational semiotics, human-computer interaction and software engineering, to recover requirements for systems for which source code and or documentation is no longer available.

Two issues arose in discussing this talk. First came the question of whether it is better (more efficient, cost-effective and/or accurate) to analyse source code or to examine user and system behaviour. Indeed, some people argued that there were circumstances where only code analysis is feasible, given time and other constraints. By contrast, it is the contention of AMBOLS that there are situations where only behavioural analysis is feasible (as source code has been lost). Again, the two views do not seem irreconcilable, though there may be a preference of one method over the other - at Stafford, there seemed a fairly clear preference for code analysis.

The second issue that arose (somewhat related to the first) concerned the applicability of AMBOLS. Several participants felt that the method was very useful, but only on a particular range of systems (in particular those which are fairly interactive and without too many complex algorithms or calculations), and that it would be helpful for these to be clearly identified[3]. It was also suggested that it would be useful in general for computer scientists to identify the range of systems their methods can work with, although in the SEBPC legacy workshops we have often experienced the difficulties in trying to form general taxonomies of solutions and persuading computer scientists to fit their work into one 'box' in any given model (Ramage and Munro, 2000).

Issues discussed at the Oxford workshop around software management were somewhat different. We agreed that, as one participant put it, “all systems (should) retire & die”, implying a need both for planned retirement and for health-checks (technical, functional and documentation). The issue of documentation, paralleling the situation foreseen by AMBOLS, also arose: “missing or inconsistent documentation is still a major issue when dealing with legacy systems, directly proportional to the size and the age of the system”. With all these issues taken together, we might look towards a perpetual maintenance/evolution process, continually monitoring the existing state of a system to stop it becoming prone to future legacy problems.

Organisational and people issues

There was a lot of discussion at the both workshops about people issues, especially around training and education. There seems very often to be a discrepancy between what is taught in universities and what is demanded by industry - participants from each of the countries represented (Portugal, UK, USA, Australia and Ireland) all reported this experience in different ways in their countries. The nature of the discrepancy is different for different countries, however.

In most countries, students want to learn the latest technologies and academics want to teach those. Participants from Portugal reported that there is a big (unmet) need by industry for skills in older technologies - our general example was COBOL programming - yet most universities teach Java and C++ programming. In the US, by contrast, industry's need is mostly for Java (etc.) skills, yet universities can't produce enough, as computer science is not a fashionable subject of study. Of course, there is an ongoing need for COBOL programmers in the US also (it's a massive industry), but few

[3] In fact this would be helpful for *all* the SEBPC projects; it simply happened that we were discussing AMBOLS when the issue arose. It is not suggested that AMBOLS is unusual in this respect.

universities teach it - one participant came from a university which *does* teach COBOL, but is under constant threat from their accrediting body of being required to drop the course.

We are in danger in computer science of forgetting our heritage - the trend so often when new tools, methods or languages arise is to throw out all the old ones, the fashion dynamic that is one of the chief causes of perceived legacy. Yet often that legacy continues both to be useful and instructive. This is especially so with skills like COBOL programming. There are still many billions of lines of COBOL code in the world, on active systems which are in use and in many cases still maintained. Yet COBOL programming is seldom taught alongside the skills of modern software engineering, and little opportunity is taken to reappraise old COBOL code in the light of modern practice for didactic purposes. This leads to a clear mismatch between past and present skill bases, and underplays the value of the existing COBOL systems to most economies. More generally, it ought to be useful to look at the skill requirements of local economies before designing university curricula; there is little evidence that this goes on in most areas.

There is a large pool of people in the workforce with skills on older technologies and languages. Should we regard these as 'legacy skills', or their holders as 'legacy people'? The term might seem offensive and dehumanising, but should at least be examined. We concluded in the workshop that the term 'legacy people' was fairly meaningless, given the continued usefulness of older skills to modern situations. However, the way in which people learn(ed) their skills is very important. If they learned them semi-mechanically, with little concern for the underlying principles involved, then it is hard to see that there is a true understanding of the skills, and as the need for them changes, the people will have to go through very time-consuming and difficult retraining - and it might be said that these are 'legacy people'. By contrast, those who learned their skills through a process that does involve looking at underlying principles will be much better off when technology or the need for particular skill bases changes and retraining becomes necessary. This divergence is often described elsewhere as being that between training and education.

Of course, the nature and breadth/depth of the principles that can usefully be learned is open to some question. Various participants reported an ongoing debate in Computer Science departments in their countries between software engineers (with a more practical orientation) and theoretical computer scientists (with a more mathematical orientation). In particular, the latter group like to see themselves as representing the underlying principles relevant to computing, and undoubtedly they do hold some of those principles, though depending on how widely one draws the issue, other theoretical discourses (especially those lying behind information systems research, such as management and sociology) may be relevant.

We talked about whether there really is a skills shortage, or if it is to do with inappropriate (too narrow) perceptions by industry about what constitutes the right skills. Most participants confirmed that in their country there was a noticeable skills shortage, especially for the latest languages and techniques. This is partly due to the way training is viewed, though - many companies do not want to train people in new technologies, only to have them leave and take their skill elsewhere. Rather, they prefer to take people that have been trained in those technologies elsewhere! The attitude is widespread, but is clearly self-defeating for the industry. However, we came to few conclusions about how to assist it.

Finally, at both workshops there was considerable mention of the need for effective knowledge management. For some participants in Oxford, organisational processes are “too dependent on (certain) people”, leading to a need for knowledge transfer. In a slightly different way, “maintenance of knowledge on large scale and over several years can only be human-controlled: investing in people returns in terms of system quality”.

Conclusions

I have tried in this paper to show the range of issues discussed in two SEBPC workshops with a rather wider range of people involved than usual. This different set of people led to a new set of perspectives being discussed from previous workshops. The discussion leaves SEBPC, and future work on legacy systems, with lots of challenges:

- What to do with people with 'legacy skills'?
- If the concept of legacy systems is no longer useful, how can we instead motivate academics and industry to look at important-but-hard-to-maintain socio-technical systems?
- Is software engineering education failing industry? If so, how can it be made better?
- How can we form objective measures of 'legacyness' (of software systems in a given context)? How can we determine it is more efficient for software to continue to be maintained, or whether the time has come to throw it away and start again?
- Is there any way of forming a useful taxonomy of partial solutions to legacy systems (formed within SEBPC and elsewhere), in such a way that practitioners can easily tell which solution is more or less appropriate?
- What is the enabling infrastructure for effective co-evolution of software and organisations, to prevent future legacy systems from happening?

These are questions which go to the heart of the SEBPC programme, and which to an extent have been answered during the programme so far. But we are still a long way from having definitive answers to any of them. That they have arisen is by no means a sign of any failure in the programme - on the contrary, it is a sign that while it has answered many questions concerning legacy systems, many more have arisen from our discussions between ourselves and with colleagues from other countries, which require further research.

Acknowledgements

Most of the ideas in this chapter are somebody else's: a big thank-you to the participants in the two workshops. Malcolm Munro co-organises the legacy workshop series with me (and came up with the five-points concept that helped the Oxford workshop to run smoothly). Thanks to Maria Papaefthimiou for discussions about the nature of this chapter. The legacy workshops, and the SABA project through which they are organised, are funded by the Engineering and Physical Sciences Research Council, through the Systems Engineering for Business Process Change programme.

Participants at the workshops

This list includes the majority of participants at the two workshops, although a few participants' names were not recorded. Participants acted in a personal capacity rather than as representatives of their organisations.

Keith Bennett (Durham University)
Stephen Cook (University of Reading)
Charles Ford (University of Arkansas)
Ian Gorton (CSIRO/University of Sydney)
Norihide Hattori (Durham University)
Peter Henderson (Southampton University)
Sue Kelly (Manchester Business School)
John Krogstie (Andersen Consulting)
Antony Lauder (University of Kent at Canterbury)
Xingkun Liu (Cardiff University)
Manny Lehman (Imperial College)
Tim Millea (Loughborough University)
Maria Papaefthimiou (London School of Economics)
Zubair Qureshi (Staffordshire University)
Magnus Ramage (Durham University)
Juan Ramil (Imperial College)
Paul Rayson (Lancaster University)
Claudio Riva (Nokia Research Center)
Bernadette Sharp (Staffordshire University)
Jianhua Shao (Cardiff University)
Perdita Stevens (Edinburgh University)

References

Alderson, Albert and Hanifa Shah (1999). “Viewpoints on legacy systems”, *Communications of the ACM* 42(3):115-116.

Bennett, Keith and Rajlich, Vaclav (2000). "Software Maintenance and Evolution: A Roadmap" in *The Future of Software Engineering*, ed. A. Finkelstein, New York: ACM Press.

Edwards, John et al. (1999). *Proceedings of the workshop on Software and Organisation Co-Evolution, SOCE99* (31 August 1999, Oxford). Loughborough, UK: University of Loughborough.

Greenbaum, Joan and Morten Kyng (eds.) (1991). *Design at Work*. Hillsdale, NJ, USA: Lawrence Erlbaum.

Kauffman, Stuart (1995). *At Home in the Universe: The Search for Laws of Self-Organisation and Complexity*. New York: Oxford University Press.

Lauder, Anthony and Stuart Kent (2000). “Legacy System Anti-Patterns and a Pattern-Oriented Migration Response” in *Systems Engineering for Business Process Change*, ed. P. Henderson, London: Springer-Verlag.

Lehman, Manny, Juan Ramil and Paul Wernick (2000). “Metrics-Based Process Modelling With Illustrations From The FEAST/1 Project” in *Systems Modelling for Business Process Improvement*, eds. David Bustard, Peter Kawalek and Mark Norris, Norwood MA, USA: Artech House.

Mitleton-Kelly, Eve and Maria-Christiana Papaefthimiou (2000). “Co-Evolution and an Enabling Infrastructure: A Solution to Legacy?” in *Systems Engineering for Business Process Change*, ed. P. Henderson, London: Springer-Verlag.

Ramage, Magnus and Malcolm Munro (2000). “It's not just about old software: a wider view of legacy systems” in *Systems Engineering for Business Process Change*, ed. P. Henderson, London: Springer-Verlag.

Stacey, Ralph (1996). *Complexity and Creativity in Organisations.* San Francisco: Berrett-Koehler.

Trist, Eric (1992). "Introduction to Volume II", in “The Social Engagement of Social Science: A Tavistock Anthology - Volume II: The Socio-Technical Perspective”, eds. E.Trist, H.Murray, B.Trist, Philadelphia: University of Pennsylvania Press, pp 36-60.