# Investigating the feasibility and worth of migrating legacy systems

Patrick Naish
pn3g10@zepler.net

**Abstract**—The abstract goes here.

## 1 Introduction

For at least forty years, mainframe computing has been the backbone of technological industry, to a greater or lesser extent. Despite having fallen out of favour in more recent years, in deference to the modern client-server style of computational interaction, mainframe software still underpins a vast amount of modern systems. There were an estimated 200 billion lines of COBOL in use in 2009[1], with no indication of a decline in its use. However, COBOL is widely considered to be a dead language, and the fact that young computer scientists are taught languages such as Java and C++ means that many are unwilling to learn older languages such as COBOL or PL/I. Those developers who do know the languages required to maintain legacy mainframe systems are rapidly reaching the age of retirement and leaving the industry, making the continued use of mainframe systems in their current form impractical, or eventually even impossible.

One of the major challenges in redevelopment is noted by Sneed[1] in that, when replacing an application with which users have grown familiar, they expect the exact same functionality and features (and potentially new features on top of those). Because of this, he says, "...it is much more difficult to replace an existing application than it is to develop one from scratch."[1]. Unfortunately, there is often a lack of resources (monetary and otherwise) available for redevelopment efforts. A 1996 case study[2] estimated that the cost of redeveloping the client's existing systems from scratch would cost approximately $24 million, over four years (with twenty staff), totalling approximately "...100 person years of effort..."[2]. It is therefore evident that there is a need for a means by which to ease the modernisation of legacy systems, without incurring unfeasibly large costs to the maintainers of said systems.

1. http://fcw.com/articles/2009/07/13/tech-cobol-turns-50.aspx

This paper will discuss the current available approaches to modernisation in section 2, the issues which must still be addressed in the field in section 3, and draw a conclusion in section 4.

## 2 Approaches to modernisation

Almonaies et al.[3] suggest four main approaches towards the modernisation of legacy systems:

- **Replacement**, in which one either rewrites existing applications in a modern language and style, or replaces them altogether (often "...with an off the shelf product..."[3]).
- **Reengineering**, in which reverse-engineering is used to add modern functionality into the legacy systems themselves.
- **Wrapping**, in which an interface is developed to encapsulate the legacy components so that they may be accessed as services from other components.
- **Migration**, in which the legacy system is transitioned into a new environment "...while preserving the original systemś data and functionality..."[3].

Sneed[4] discusses this in terms of migration versus integration, where migration involves the actual transferral and/or transformation of code and components into a new environment. Integration instead refers to remotely accessing components from the new environment, without transitioning existing components into it.

As previously noted in 1, replacement is potentially either prohibitively expensive[2] or fails due to difficulties users experience in adopting the new software[1]. This is not to say it is never a practical option; for smaller-scale applications, replacement may well be the most cost-effective solution after analysis. For the most part, however, existing literature deals with migration[1], [2], [4]–[15], with elements of wrapping[9], [15], [16].

The CelLEST project[17] is discussed in the context of reengineering, although this still makes heavy use of wrapping.

## 2.1 Service-Oriented Architectures

The most popular architecture to which legacy systems are migrated is that of a **Service-Oriented Architecture** (SOA)[3], [4], [7], [9], [12], [18]. A service is defined by Perrey and Lycett[19] as "...the boundary between a consumer understandable value proposition and a technical implementation...". That is to say, it provides an interface between an agent and some functionality to allow for interaction without necessitating a deep understanding of underlying code. SOA is, therefore, a design pattern based on the interaction between these services.

Some of the advantages of SOA for the purposes of migrating legacy systems are its flexibility, reusability and abstraction[3]. As Aversano et al. note[5], the mainframe systems were largely built before the internet became prevalent, using a centralised architecture to which a limited number of trusted users would connect. In the modern world, systems are accessed far more widely (e.g. through the internet), by a large number of users, and therefore the architecture must evolve accordingly.

## 2.2 Tools for code analysis

One of the main tasks in migrating systems is to analyse code for elements which can be extracted and encapsulated, as well as judging the feasibility of migrating a certain piece of code. This would be prohibitively complex (and therefore time-consuming and costly) for a human to perform manually, so tools have been developed to do so automatically (as far as possible).

Sneed[4] uses the **COBAudit** tool to assess a large volume of programs in his pilot project. This took "...56 different size measurements..."[4] from each program, including lines of code and number of data structures. From these, eight metrics were used to determine the programs' overall complexities (as discussed by Sneed in a 1995 paper[20]). These complexity measurements are then normalised using **SoftAudit**, with values above 0.5 indicating high complexity, and those below indicating low complexity. Any values over 0.8 can be considered to indicate extreme complexity. From the normalised results, it is then possible to select programs which are suitable for reuse, as well as identify the sections of them for which this will be most straightforward or difficult.

Lewis et al.[21] use the **Understand for C++**[2] to extract both a static analysis of a C++ application, and metrics for assessing the ease of migration. They then use the **Architecture Reconstruction and Mining** (ARMIN)[22] tool to attempt to reconstruct the architecture of analysed programs, in order to find inter-service dependencies (as well as issues with these dependencies in their current form). This analysis is then used to create a migration strategy moving forward.

## 2.3 Tools for migration

Once reusable code has been identified, there are tools available for facilitating migration. One set of these tools is **COB2WEB**, created by Sneed[12], comprising **COBStrip**, **COBWrap**, and **COBLink**, as well as the **COBAudit** tool mentioned in subsection 2.2.

### *2.3.1 COBStrip*

This tool uses code slicing to decompose programs. This concept is discussed in some de-

2. http://www.scitools.com/

tail by Weiser[23], and applied practically in the context of migration by Sneed[4], [12]. Its purpose is to separate out various 'slices' from a program, where each slice represents a path through the program that will perform a given function of the original code. These slices are used to "...generate multiple instances of the same program..."[4], [12], which can then be run as individual web services. The tool requires some user interaction, but reduces the workload of slicing significantly.

#### *2.3.2 COBWrap*

This tool uses the stripped programs generated by **COBStrip**, and replaces input/output (I/O) operations with "...calls to a wrapper module..."[4], [12], as well as moving required data to the correct sections of the program. This can be performed without any user interaction, as it is a reasonably simple operation when used with predictable programmatic structures (such as those in COBOL).

#### *2.3.3 COBLink*

This tool uses the wrapped programs generated by **COBWrap**, and creates **Web Services Description Language** (WSDL)[3] interfaces for both incoming requests and outgoing responses, as well as COBOL modules for translating between the WSDL and COBOL parameters. This process has been described by Sneed in detail[24], as well as applied in a practical application[4], [12]. Once these are generated, it is possible to interact with the services with **Simple Object Access Protocol** (SOAP)[4] messages, thereby accessing the functionality of the legacy application through a modern, flexible interface.

### 2.4 Techniques and approaches

#### *2.4.1 Service-Oriented Migration and Reuse Technique (SMART)*

SMART was described by Lewis et al. as a "...process for evaluating legacy components for their potential to become services..."[21], [25]. As opposed to the tools mentioned in subsection 2.2 and subsection 2.3, this is a set of in-depth analyses of various aspects of the legacy system, target environment, user base and required effort. This is something more of a software engineering approach than a computer science one, as it involves considering the business as a whole and developing strategies, rather than only considering the technical challenge involved. This, of course, leads to a larger amount of work needing to be performed initially, but creates a more cohesive strategy for the overall migration process.

Part of this process may also involve architecture reconstruction, using a tool such as **ARMIN** (previously mentioned in subsection 2.2). This allows for analysis of **Rigi Standard Format** (RSF)[5] files, containing metadata extracted from source files in various languages[22]. These analyses are used to create models and abstractions which, in turn, "...produce higher-levels[*sic*] models and views..."[22]. Eventually, through this process, the "...desired set of views is produced."[22] An evident use of this is to separate out a legacy application into **Model–View–Controller** (MVC) layers, as additionally discussed by Bodhuin et al.[6].

#### *2.4.2 Service-Oriented Software Reengineering Methodology (SoSR)*

#### *2.4.3 Ubiquitous Web Applications Design Framework (UWA)*

#### *2.4.4 CelLEST*

The CelLEST project[6]

3. http://www.w3.org/TR/wsdl
4. http://www.w3.org/TR/soap
5. http://www.rigi.cs.uvic.ca/downloads/rigi/doc/node52.html
6. http://webdocs.cs.ualberta.ca/~stroulia/CELLEST/new/

### 2.5 Comparison of approaches

## 3 Issues which must be addressed

### 3.1 Cost-effectiveness

### 3.2 Maintainability

### 3.3 Risk

### 3.4 Automation

### 3.5 Testing

## 4 Conclusion

## Acknowledgements

The author would like to thank Prof. Michael Butler for his supervision during the writing of this paper.

## References

[1] H. Sneed, "Migrating from COBOL to Java," *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pp. 1–7, 2010. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5609583

[2] D. G. Duncan and S. B. Lele, "Converting from mainframe to Client/Server at Telogy Inc," *Journal of Software Maintenance-Research and Practice*, vol. 8, no. 5, pp. 321–344, 1996. [Online]. Available: http://dx.doi.org/10.1002/(SICI)1096-908X(199609)8:5⟨321::AID-SMR135⟩3.0.CO;2-4

[3] A. Almonaies, J. Cordy, and T. Dean, "Legacy system evolution towards service-oriented architecture," in *International Workshop on SOA Migration and Evolution (SOAME 2010)*, 2010, pp. 53–62. [Online]. Available: http://research.cs.queensu.ca/home/cordy/Papers/ACD_MigToSOA_SOAME10.pdf

[4] H. M. Sneed, "A pilot project for migrating COBOL code to web services," pp. 441–451, 2009.

[5] L. Aversano, G. Canfora, A. Cimitile, and A. D. Lucia, "Migrating legacy systems to the Web: an experience report," *Proceedings Fifth European Conference on Software Maintenance and Reengineering*, 2001.

[6] T. Bodhuin, E. Guardabascio, and M. Tortorella, "Migrating COBOL systems to the Web by using the MVC design pattern," *Ninth Working Conference on Reverse Engineering, 2002. Proceedings.*, 2002.

[7] G. Canfora, A. Fasolino, G. Frattolillo, and P. Tramontana, "Migrating interactive legacy systems to Web services," *Conference on Software Maintenance and Reengineering (CSMR'06)*, 2006.

[8] G. Canfora, A. Cimitile, A. De Lucia, and G. A. Di Lucca, "Decomposing legacy programs: A first step towards migrating to client-server platforms," *Journal of Systems and Software*, vol. 54, pp. 99–110, 2000.

[9] G. Canfora, A. R. Fasolino, G. Frattolillo, and P. Tramontana, "A wrapping approach for migrating legacy system interactive functionalities to Service Oriented Architectures," *Journal of Systems and Software*, vol. 81, pp. 463–480, 2008.

[10] A. D. Lucia, G. D. Lucca, A. Fasolino, P. Guerra, and S. Petruzzelli, "Migrating legacy systems towards object-oriented platforms," *Proceedings International Conference on Software Maintenance*, 1997.

[11] G. Lewis, E. Morris, and D. Smith, "Analyzing the reuse potential of migrating legacy components to a service-oriented architecture," *Conference on Software Maintenance and Reengineering (CSMR'06)*, 2006.

[12] H. Sneed, "COB2WEB a toolset for migrating to web services," *2008 10th International Symposium on Web Site Evolution*, 2008.

[13] H. M. Sneed and K. Erdoes, "Migrating AS400-COBOL to Java: A Report from the Field," in *2013 17th European Conference on Software Maintenance and Reengineering*, 2013, pp. 231–240. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6498471

[14] L. W. L. Wu, H. Sahraoui, and P. Valtchev, "Coping with legacy system migration complexity," *10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05)*, 2005.

[15] H. Sneed, "Encapsulating legacy software for use in client/server systems," *Reverse Engineering, 1996., Proceedings of the ...*, pp. 104–119, 1996. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=558885

[16] C.-C. Chiang, "Wrapping legacy systems for use in heterogeneous computing environments," pp. 497–507, 2001.

[17] E. Stroulia, M. El-Ramly, and P. Sorenson, "From legacy to Web through interaction modeling," *Software Maintenance, 2002. Proceedings. International Conference on*, pp. 320–329, 2002.

[18] A. Koschel, C. Kleiner, and I. Astrova, "Mainframe Application Modernization Based on Service-Oriented Architecture," *2009 Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns*, pp. 298–301, Nov. 2009. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5359601

[19] R. Perrey and M. Lycett, "Service-oriented architecture," *Applications and the Internet Workshops, 2003. Proceedings. 2003 Symposium on*, pp. 116–119, 2003. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1210138

[20] H. Sneed, "Understanding software through numbers: A metric based approach to program comprehension," *Journal of Software Maintenance: Research and Practice*, vol. 7, pp. 405–419, 1995. [Online]. Available: http://www3.interscience.wiley.com/journal/113446591/abstract

[21] G. Lewis, E. Morris, L. O'Brien, D. Smith, and L. Wrage, "SMART: The service-oriented migration and reuse technique," Tech. Rep. September, 2005. [Online]. Available: http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA441900

[22] L. O'Brien, D. Smith, and G. Lewis, "Supporting Migration to Services using Software Architecture Reconstruction," *13th IEEE International Workshop on Software Technology and Engineering Practice (STEP'05)*, pp. 81–91, 2005. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1691635

[23] M. Weiser, "Program Slicing," *IEEE Transactions on Software Engineering*, vol. SE-10, 1984.

[24] H. Sneed, "Wrapping legacy COBOL programs behind an XML-interface," *Reverse Engineering, 2001. Proceedings. Eighth Working Conference on*, pp. 189–197, 2001.

[25] G. Lewis, E. Morris, and D. Smith, "Service-Oriented Migration and Reuse Technique (SMART)," *13th IEEE International Workshop on Software Technology and Engineering Practice (STEP'05)*, pp. 222–229, 2005.