Eating an Elephant Iterative Maintenance & Modernization of a Legacy Software System

Noah C. Spahn
Software Engineer: Enterprise Technology Services
University of California, Santa Barbara
noah.spahn@ucsb.edu

***Abstract*—This paper presents a survey of academic software models which led to the development of new model for maintaining a legacy enterprise software system whilst modernizing it towards a migration. This paper seeks to inform software engineers. Since informed engineers are best equipped to combat the entropic decline of an aging system. The specific points emphasized may not translate into the modernization of most legacy projects, however cataloging the journey may be helpful for others who are about to embark upon a similar trek.**

I. Introduction

The fact that systems in use must be maintained in order to keep in step with usage requirements has already been well documented. There is no shortage of literature on software maintenance. A more recent addition to the corpus of literature on software engineering is that which provides guidance and direction on the practice of migrating from a legacy system to a new software system that better meets the needs of the end users. Even here, there is no shortage of literature on the subject. However, it has been noted that the proposed models which address the topic are largely specific to a portion of the practice: either the planning or the implementation. This section of the paper catalogues the academic landmarks which led to the development of the models used in the iterative maintenance of an enterprise legacy application at the University of California, Santa Barbara (UCSB).

A. Maintenance Models

There are many well known and widely accepted maintenance models for the software engineering lifecycle. These models are indispensable to any large scale software organization producing mission critical software (where lives are at stake). The standards fall short of what can be deemed as a practical level of rigor for a smaller software organization with less resources at their disposal[1].

The ISO 14764 standard depicts software migration as a departure from the cycle of software maintenance. This is simply not an option for a smaller team without the resources to devote to separate maintenance and (migration) or development efforts.[2]

B. Structured Migration Process

Many organizations gravitate towards wholesale migration away from a legacy software system. The Structured Legacy to SOA migration process and it's evaluation in practice [3] combines the planning and implementation phases to provide a complete model which coalesces both parts into one.

Not all the phases were common to all migration efforts. The omission of the 'migration feasibility determination phase' was common among organizations. This glaring omission stood out as a potential area for further investigation,.

C. Success Factors Model

Equipped with a clear understanding of the entire migration process brings the weight of responsibility. Subsequently the question would be posited: "How would an organization ensure a successful migration?", or more specifically: "what are the common factors of successful migration efforts?". Galinium and Shabbaz [4] conducted a systematic survey of the published case studies on the migration of legacy software systems. The study indicated a strong correlation between migration success and the involvement of the heterogeneous group made up of both technical and business stakeholders.

D. Iterative Model for Migrating Legacy Systems

With the findings of these three papers as guideposts, the ingredients were on the table to cook up a new model to equip an under-staffed software organization fo proceed with an informed, well planned and intentionally executed migration of a legacy software system to a service oriented architecture . The model was drafted as the Iterative Model for Migrating Legacy Systems (IMMLS) and put into practice by a small team at the University of California, Santa Barbara[5].

A year of iterating through the IMMLS surfaced some incompatibilities. The model did not devote enough time and effort to the daily ongoing maintenance of the legacy system. A small team which handles development, operation and support cannot forfeit support in lieu of development. The software engineers were torn between the rigor of phased migration and the distraction of change requests. Clearly, the adjustments were necessary.

E. eXtreme Programming as Maintenance

Choudhari and Suman propose a model that catalogs change requests as either bug reports or system enhancements. The requests are summarized as stories (brief descriptions) and assigned effort estimations. These stories feed into the extreme programming model, which emphasizes respect and communication[6].

II. Modernization as Migration

Considering the need to incorporate change requests, the IMMLS became a maintenance model with a clear intent to modernize towards migration[7].

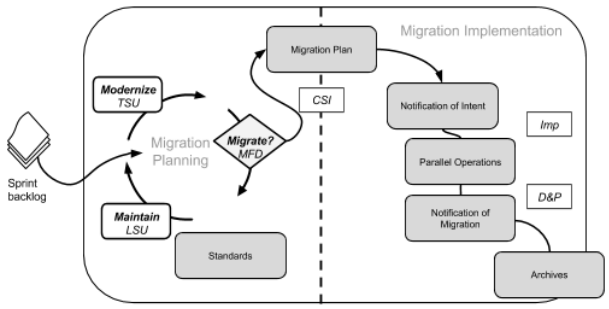

Fig. 1: Iterative Model for the Migration of Legacy Systems [7]

We can follow this model from left to right, but most of the model takes place on the left-hand side, cycling over the migration feasibility determination phase (AKA, migration planning). Requests for change feed the process of maintenance. The change requests can come from any stakeholder. Requests are written in a concise language known as a story. Stories are written on an index card, written in a style that lacks ambiguity and clearly defines a deliverable. Written stories are evaluated and given an effort estimation before work begins, then the story is placed in the sprint backlog.

When a software iteration is planned, items are drawn from the ordered backlog and placed into the sprint. Each correction that is made to a legacy system requires an understanding of the system (to keep from breaking existing functionality). Likewise, even enhancements can not be properly added in without an understanding of the current system. Enhancements also provide the added benefit of clarifying the understanding (and vision for) the target system.

The agile movement has shown us the value of a shortened feedback loop [8], and we can make use of a shorted feedback loop from within each of the phases of the structured migration. Specifically, we propose taking the small steps in the form of the phases described in the structured six-phase process.

The idea being that by aggressively maintaining the legacy application and extending the usage of it, the inevitability of wholesale migration is postponed as the application becomes better suited for migration.

III. Conclusion

The desire to discard a legacy system and migrate to a new framework led to the investigation of legacy system migration. An amalgamation of academic models provided a promising framework for such a migration. The practical application of the model surfaced several pain points for the pilot organization, and adjustments were made. The adaptation of the model has led to some significant re-workings of a legacy application and several modules have been replaced entirely.

A. Problems

One problem with the model is the lack of a definitive end product. Once the focus was shifted from wholesale legacy system migration to an iterative enhancement model with modular migration components the final goal became ambiguous. The legacy system is constantly being re-evaluated and re-worked, with the underlying understanding that a system that is in use will require constant and vigilant maintenance to keep in favor with end users. Every system has technical debt and legacy parts, the focus has shifted from removing the legacy system to removing the legacy parts of the system. There will always be legacy parts in a system, and hence the job is never done.

Another problem is the challenge of metrics. Assuredly, we cannot improve that which is not measured [9]. We have not settled on a satisfactory method of capturing the 'right amount' of data for the processes in place. The crux of the problem is to find a balance where meaningful metrics are collected on the processes (of development, maintenance and support) but the practice of gathering metrics does not impede the actual processes.

B. Future Work

There is still work to be done to realize the potential (and deficiencies) in this model. There are two main areas which are ripe for further exploration: the continued use of the model to a definitive 'end' so that a retrospective can be conducted and the repeated application of the model within different contexts. The former is stymied by the challenge of software entropy (a system in use is never complete) and the latter will be dependent upon the adoption of the model by others.

References

[1] P. Grubb and A. A. Takang, *Software Maintenance: Concepts and Practice*. World Scientific, 2nd ed., 2003.

[2] International Standards Organisation (ISO), *Standard 14764 on Software Engineering - Software Maintenance*. ISO/IEC, 2006.

[3] R. Khadka, A. Saeidi, S. Jansen, and J. Hage, "A structured legacy to soa migration process and its evaluation in practice.," in *MESOCA* (A. D. Ionita, G. A. Lewis, and M. Litoiu, eds.), pp. 2–11, IEEE, 2013.

[4] M. Galinium and N. Shahbaz, "Success factors model: Case studies in the migration of legacy systems to service oriented architecture," in *Computer Science and Software Engineering (JCSSE), 2012 International Joint Conference on*, pp. 236 – 241, IEEE, 2012.

[5] N. Spahn, "Migration planning - iterative model for migrating legacy systems." unpublished project paper, 2016.

[6] J. Choudhari and U. Suman, "Extended iterative maintenance life cycle using extreme programming.," *ACM SIGSOFT Software Engineering Notes*, vol. 39, no. 1, pp. 1–12, 2014.

[7] N. Spahn, "When can we migrate? a model for approaching legacy system migration." https://hal.archives-ouvertes.fr/hal-01687747. [Online; accessed 10-August-2018].

[8] J. Highsmith, *Agile Project Management - Creating Innovative Products*. Boston: Pearson Education, 2004.

[9] W. S. Humphrey, *Managing the Software Process*. SEI Series in Software Engineering, Addison-Wesley, 1989.