Identifying Problems in Legacy Software Preliminary Findings of the ARRIBA Project ⋆

Isabel Michiels[1], Dirk Deridder[1], Herman Tromp[2], and Andy Zaidman[3]

[1] Programming Technology Lab, Vrije Universiteit Brussel
Pleinlaan, 2 , 1050 Brussel, Belgium
{Isabel.Michiels,Dirk.Deridder}@vub.ac.be
[2] Universiteit Gent, INTEC, Sint-Pietersnieuwstraat 41
9000 Gent, Belgium - Herman.Tromp@UGent.be
[3] Universiteit Antwerpen, Lab On Re-Engineering, Middelheimlaan 1
B-2020 Antwerpen, Belgium - Andy.Zaidman@ua.ac.be

Abstract. The goal of this experience report is to identify some of the key problems of today's enterprises that have to deal with managing their large business critical software systems. Our motivation to do so is based on preliminary findings from the ARRIBA project. The work we present here form our preliminary conclusions of the first 6 months of the project, where we visited some of these enterprises, to identify their main needs of today.

1 Introduction

The dynamics of modern business applications is characterized by a constant need for integration and restructuring and this at a much larger scale than ever before. This is often driven by the physical integration and restructuring of companies, which consequently results in a need to alter their ICT infrastructures to accomodate the changed business activities. Possible examples are the redefinition of a corporate strategy, a corporate take-over, a conversion of the existing infrastructure from a data processing model towards a service oriented model, etc ...
This continuous modification process will finally result in a situation where several software systems have to collaborate in a way that was never (or could never have been) anticipated in their original design.
Such large-scale software applications are often referred to as *legacy applications*. In this report we will adhere to the following definition of a legacy application [7]:

> *A legacy system is an operational system that has been designed, implemented and installed in a radically different environment than imposed by the current ICT strategy*

When burdened with the task to enable the collaboration of these separate systems, having access to a rich collection of documentation (preferably also feedback from the original designers/ programmers of the system) is imperative. Unfortunately, in all but

⋆ This research is funded by the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT)

a few cases, this documentation remains non-existing or has become completely out of date due to evolution of the software system. Generally speaking, one could state that the only true description of the information structures and the implemented behaviour is locked up in the running software system itself. Hence to obtain a sufficient (active) set of documentation one will have to turn to analysing the available static (e.g. source code, data models) and dynamic (e.g. runtime event traces) artifacts of the system.
There are five ways to handle a legacy situation in which a change is imposed :

1. Rewrite the source code from scratch
2. Refactor (rewrite the code, but in a different context)
3. Migrate to a more desirable target system
4. Reuse a part of the legacy system
5. Combine 1,2,3 and 4

As we have seen in our first findings, option 5 is most commonly used.
In this report we provide a preliminary overview of a number of encountered problems when confronted with legacy software. We have based ourselves on the results of visiting three major Belgian enterprises in the context of a research project called ARRIBA. In the following section, we will briefly describe the ARRIBA project. Then we will present our first findings in section 3 and in section 4 we will point out future work for the project. We will then round up in section 5 with our conclusion.

2 The ARRIBA project

The ARRIBA project [1] is a generic research project funded by the IWT, Flanders [2]. The project started in October 2002 and will allow 6 researchers to work on the project for 4 years [1].
The aim of ARRIBA is to provide a methodology and its associated lightweight tools in order to support the integration of disparate business applications that have not necessarily been designed to coexist. Inspiration comes from real concerns that are the result of an investigative effort on the part of some of the research partners in this consortium. The object of this investigation is the identification of mainstream ICT problems within a representative forum of Belgian enterprises (large and small) that rely on information technology for their critical business activities. Part of what we propose to investigate is covered by the newly named discipline of Enterprise Application Integration (EAI); another part is covered by re(verse) engineering; however, our ambitions reach further.
At the roots of the ARRIBA project are two driving forces. On the one hand we have a consortium of research groups that have been active in the field of software engineering and more particularly in re(verse) engineering, software evolution and software architectures[2]. These groups have a fairly long-standing history of cooperation and they feel

[1] ARRIBA: Architectural Resources for the Restructuring and Integration of Business Applications, see http://arriba.vub.ac.be

[2] There are 3 Flemish academic partners involved: Vrije Universiteit Brussel (VUB), Universiteit Gent (UG) and the Universiteit Antwerpen (UA) and two other European partners, UCL in Louvan-La-Neuve, Belgium and SCG, Berne, Switzerland. The latter two play a supporting role

confident that they can join forces and tackle the new and ambitious problem domain targeted in the ARRIBA proposal.
On the other hand, we have the already mentioned and recently created forum of Belgian enterprises interested in a joint initiative to identify generic problems and likewise generic solutions plaguing their ICT component[3]. This forum has the form of a foundation hosted by what could best be described as a collective spin-off of the five Flemish computer science departments.
These two contributing agents; the first providing the content of the ARRIBA project, the second providing the context; will guarantee the correct identification of the problem setting, the marshalling of the required scientific and technical resources and the proper channeling of the results to the business world.
The user committee of the ARRIBA project currently consists of 7 Belgian enterprises. They form the steering group of the project: they regularly check if we tackle current ICT problems and during the evolution of the project they will see whether our results will be industrially applicable. The next section reports on the first findings based on visits of part of the user committee members.

3 First Findings

During the first six months of ARRIBA, we visited 3 major Belgian enterprises: the KBC group[4], Banksys [5] and LCM [7] [6]. As preparation for these visits, we prepared a question list according to [4]. One of these visits was organized in a workshop format, while the other two were more Q&A sessions based on presentations by the companies In a later phase, other visits to other companies are planned.
In what follows, we have organized our findings into common themes:

The Mainframe Syndrome

All of these organisations depend heavily on proven technology and duplicate datacenters for their critical business activities; this is an environment strictly used for controlling processes to be able to ensure operational performance. UNIX-like systems are also used, but only in more peripheral and less critical applications. They are considered to be less reliable, and therefore unfit to support their essential business operations. This situation has grown due to historical circumstances, but it indicates that there is a serious resilience towards new and not yet proven technology (hardware as well as software). On the other hand, there is a willingness to install UNIX-like systems, but integration with the existing mainframe environment remains a very big issue [6]. Previous efforts

[3] These Belgian enterprises are grouped in a *User Committee* currently consisting of 7 companies: Inno.com, KBC, LCM, Banksys, Toyota, KAVA and Pefa

[4] KBC is a large banking company that holds three major product factories: banking, insurance and marketing activities

[5] Banksys is one of the most important providers of the infrastructure for electronic financial transactions in Belgium

[6] Landsbond der Christelijke Mutualiteiten (LCM) is a large organisation responsible for the redistribution of health care allowances, and offers also a number of related social services

to migrate to a Microsoft technology-based system have proven to be unsuccessful, at least in the case of LCM [7].

Organisation and Human Resources

Most organisations have a pretty strict and project-based organization which is clearly reflected by the Human Resources setup. This adds considerably to their latency and capacity to adapt. Take as an example LCM: they have about 200 COBOL developers, and (only) 4 or 5 Java-aware software engineers. It is clear that in such an environment there will be a lot of resistance towards new developments (the systems are functioning properly, aren't they, so why change anything?). The previous point is reflected in the structure of the organisation : separated business units, project driven work structure,.... A central authority to control major revitalisation efforts, to enforce architectural consistency and provide a deployment policy is often missing or very difficult to install.

Coding Standards and Techniques

In general, it is estimated that between 60 and 80 % of today's operational code is still written in COBOL [3]. Some C or Java code is also present, but only in small quantities. Knowledge about the systems is partly lost and only evident in the code itself, e.g. people have left the company, documentation is very poor (and out of sync with the current system) or not present, etc.... When validating the quality of these mainframe COBOL applications, usually the 80/20 rule will manifest itself: 80% of the coding problems are caused by 20% of the code (also known as *The Pareto Principle*) [5].
Regarding architectural issues, migration has been put forward as the main bottleneck of the restructuring process; however we found that companies experience that integration with new technologies is much more important (and also more difficult). Take as an example the introduction of new environments (like J2EE) for new applications: the real challenge here is how to let these connect or communicate with the other COBOL applications on the mainframe.

Data and Information

Large-scale software systems consequently also have to deal with large amounts of data. Unfortunately, in most legacy systems, the use of a Relational Database Management System (RDBMS) is scarce. Proprietary flat file systems are still in use, but migration to using a RDBMS system has received top priority.
In large organizations, a Corporate Data Model is hard to enforce. The reason for this is simple: there is no central ownership of data or information items in use by these companies. This often leads to a rapid growth of different information models, where every part of the organization has its own view on that same information, with differences in structure and even in the semantics of these information models. Take as an example the concept of a *customer*: it is interpreted differently in other business units within the company; a *customer* that buys something is very different from a *customer* that complains about the companies' delivered products. So the quality of the data models and

the data itself, because of the lack of a responsible person, is far from guaranteed. Also, a consistent view on the information between the business units themselves [8] is missing. As a consequence, migrating the software and the information models becomes a real problem.
At the KBC for example, the use of a uniform data model cannot be enforced, but instead they enforce a uniform message model. This means that they clearly specify the syntax and semantics of messages that are sent between different software applications, not the form of the data itself. In practice, this approach has proven to give some pretty good results.

Using Standard Packages

One of the possible solutions these companies bring forward to better structure their applications is using standard ERP packages. However, this causes several problems. Experience proves that packages have a strong front-end (or presentation layer), but a weak back-end for performance. On the other hand, some applications require the use of certain packages since they implement international standards.
Security forms a large problem as well; it can be a conclusive reason for refusing the use of a package. Another drawback of using packages is that they are expensive and sometimes have not enough functionality. Customizing these packages can be dangerous due to package updates; therefore a decision is made every time whether a package should be bought or written from scratch.
Another issue is that the view of the package on the business domain does not map directly to the real world, and the amount of work to be done for integrating these packages into the existing application is highly underestimated. Formulated otherwise: there is a semantic gap between the ”standard” package and its existing information model; and performing gap analysis is time-consuming.
Another open question that still remains is how to map the companies’ business process model onto the ICT infrastructure of the predefined package.

Datawarehousing

Setting up datawarehousing activities is not a trivial thing to do: project-driven businesses (like the KBC) need to set-up a project first, mainly about collecting meta-data information. Since this data is cross-cutting different business units, these projects are difficult to ’sell’, because it is difficult to find a single business case for them. After all, possible profit can only be shown after a while. Most extraction of meta-data is done by interviewing people: they are the most valuable sources of information. And although most companies see the importance of datawarehousing, it is not really clear yet what they will do with all the meta-data information.

Enterprise Application Integration (EAI)

This rather new domain dates from more or less 1996. At this moment there is a growing number of enterprises that try to use this, usually under the form of standard EAI

tools (at the KBC they use(d) eGate and Tibco). For KBC, they have been using this through a business case since 1998. Problems that arise now for KBC mainly come from handling different EAI tools at once: now it is almost impossible to go back to using only one tool throughout all units within the company. Instead, the use of the tools is being extended, according to the needs and applications, inside the growing domain of EAI. Again because of the lack of centralized responsibility, there is a proliferation of different versions of these tools.

The IT Development Process

The IT Development process is usually well-defined within a company policy and a lot of attention is paid to it. However, as mentioned before, it is not technology-driven, which has as a downside that projects that do not have a business case (that are hard to sell within the company), cannot be realized.
Developing and collecting documentation is, in some cases, part of the predefined software development process of the company. Unfortunately it is too often neglected for obvious reasons (e.g. time consuming, limited budget). So there is documentation available, but it is in most cases not up-to-date with the current software. So the source code and the information models are often the only reliable source of documentation.

4 Future Work

Based on our first findings, we conclude that the first step for restructuring legacy applications is to understand and analyze the source code (we will call this *Code Mining*). This can be accomplished by analyzing static as well as dynamic information and taking into account the data and information models as well.
The second step could then be to identify lightweight tools that can, using the results of the analysis of the first step, automatically extract architectural information, documentation or domain knowledge out of the source code and data models. A last step could then be to incorporate changes into the extracted artifacts and propagate these back into the code (forward engineering).
Future tracks will emphasize more on COBOL and its environment and on how to use dynamic information as well:

Emphasis on COBOL code and its environment Since the companies we presented before are willing to let us experiment on their code, we will concentrate in a first phase on studying COBOL code and its environment. We would like to apply some of the already known tools (that were developed in the labs of one of the academic partners), like SOUL [7] or CodeCrawler [8]. Since these tools were not developed specifically for COBOL, we first have to see how we can tweak them to use them within this context.
We have started to work on transforming COBOL into a more portable platform: we intend to use XML as a portable format for source code representation. We can then

[7] Smalltalk Open Unification Language - http://prog.vub.ac.be/research/DMP/soul/soul2.html
[8] see http://www.iam.unibe.ch/ lanza/CodeCrawler/ codecrawler.html

manipulate XML documents inside other language platforms. In a second phase (forward engineering), we could try to manipulate this XML representation (either directly on the DOM model, either through XSLT) and retransform it back to COBOL to actually restructure the code.
In the near future, we would also like to investigate in which way we can reuse techniques developed for object-oriented systems, like code metrics, code refactoring,... for restructuring (and enhancing code quality) non-OO legacy systems.

Using Dynamic Information in the context of reverse engineering, static analysis is the term used for a reengineering effort based solely on the information that can be found in the source code of the software. In many cases this analysis is computationally very intensive and doesn't give the whole picture. Dynamic analysis uses information collected during the execution of the program. The information we collect is called an event trace and consists of a list of method invocations, procedure calls, object instantiations, etc. A clear advantage of using dynamic analysis is that the information you have is always correct with respect to the execution of the program, but a clear disadvantage is the amount of information you have to wade through. Research in this direction will revolve around finding event sequences that logically belong together in the execution of the program, i.e. a clustering operation. These clusters can then be abstracted to patterns that point to key functionality in the software.

5 Conclusion

In this experience report, we have identified some of our preliminary findings of the ARRIBA project, which aims at providing lightweight methodologies and tools for the integration of software entities that have not necessarily been designed to cooperate.
During the first phase of the project we visited 3 out of 7 enterprises that are part of the project's user committee, and we presented some surprising commonalities found in their current ICT restructuring schemes.
Finally, we ended by pointing out our future work for this ARRIBA project, with as next intermediate goal to experiment with some mainframe applications (witten in COBOL) and applying some already known lightweight tools to see what we can achieve.
In the near future, we will continue with the company visits.

References

1. Arriba snelt integratie te hulp. Datanews, weekblad voor de netwerkeconomie, http://www.datanews.be, nr 04 - 31st of January, 2003.
2. Institute for the promotion of innovation by science and technology in Flanders(IWT). http://www.iwt.be.
3. G. D. Brown. Cobol: The failure that wasn't. COBOLReport.com - http://www.csis.ul.ie/COBOL/course/.
4. S. Demeyer, S. Ducasse, and O. Nierstrasz. *Object-Oriented Reengineering Patterns*. Morgan Kaufmann and DPunkt, 2002.
5. V. Pareto. The pareto principle. http://www.paretolaw.co.uk/principle.html.

6. D. Plakosh, S. Comella-Dorda, G. A. Lewis, P. R. H. Place, and R. C. Seacord. Maintaining transactional context: A model problem. Technical report, SEI, august 2001. CMU/SEI-2001-TR-012 - ESC-TR-2001-012.
7. H. Tromp and G. Hoffman. Evolution of legacy systems, strategic and technological issues, based on a case. paper also submitted to this workshop at ICSM 2003.
8. K. Vandenborre, P. Heinckiens, G. Hoffman, and H. Tromp. Coherent enterprise information modelling in practice. In *Proceedings of 13th European-Japanese Conference on Information Modeling and Knowledge Bases, Kitakyushu, Japan, June*, 2003.