Managing the Evolution of Legacy Applications

August 2015

Introduction

Maintaining legacy applications is a difficult problem that most long running businesses will eventually confront. Legacy applications are old applications have been in use for many years and are often critical to the functioning of the business. Since the business environment is always changing, there is continual pressure to support new requirements and to fix existing bugs. However, changing these applications is hard.

There are a number of reasons why maintaining legacy applications turns out to be such a thorny problem. Most of the original developers are likely to have gone on to other projects, the technologies used in the applications are no longer current, having been supplanted by newer and more enticing technologies, and nobody really understands how the application is implemented.

Regardless of whether you are looking to supplant a legacy application or to gradually evolve it to support new requirements, the key to managing the life cycle of the application is to maintain an understanding of how the application is implemented and how it is used.

Understanding the application from different perspectives

In order to understand a legacy application, we need to understand how it is used by various stakeholders. Indeed, these different perspectives are reflected in the design on the application. This is actually no different from what we do to understand the complexity of any system.

Understand how it is used. Understanding the use is critical to understanding the design. Over time special use cases are added that are unique to the business and are manifest in the particular way the application is designed and implemented. Furthermore, the nature of usage has a deep influence on performance requirements. For instance, an application backend responding to interactive use has very different requirements from a backend responding to high frequency programmatic access.

Understand how it is deployed. This is one of the most neglected aspects of architectural analysis. One reason is that in the past many applications were monoliths and so there was not much to understand. These days the application can be distributed in multiple containers and even monolithic applications are now broken up into multiple libraries. Many applications also ship with multiple third party libraries.

Understand how it is built. It is necessary to understand how each deployed artifact is built. This is particularly true for languages such as C/C++ where users specify a variety of compile time options when generating object files. These options can be used for generating different variants from the same source code. Without an understanding of these options, it wouldn't be possible to analyze the code.

Understand how it is structured. This is the area that the developer cares a lot about. A large part of the complexity resides here. The code can be organized in thousands of interdependent files and a key goal of architectural analysis is to organize these elements into modular groups. This is where architecture discovery becomes necessary. This is also where tool support is invaluable. Here are some techniques you can use:

1. Examining the existing artifacts is an excellent starting point. For instance, the file/directory structure or the package/namespace structure is already a guide to how the developers organized these code elements.

2. Apply partitioning and clustering algorithms to discover the layers and independent components. Even if the architecture has eroded, identifying the minimal set of dependencies that cause cycles will often lead to the discovery of the intended layers.
3. Experiment with what-if architectures. Create different logical modules and examine the dependencies. For example, if you are looking to componentize, then create logical components using the current set of files/classes/elements. If there are no dependencies between these components and they are intended to be independent of each other, then the componentization works. On the other hand, if there are dependencies between these components, you know what dependencies to eliminate.
4. Ask the senior developers and the architects who are involved in supporting the application. They will already have some understanding of the architecture. They can also assist in experimenting with what-if architectures. In fact, experimenting with what-if architectures will sharpen their understanding of the system.

The goal of architecture discovery is to arrive at an agreement about the organization of the application. It is one of the most effective ways to make the code understandable and maintainable and a clear understanding of the architecture will prevent further erosion. You can even make rules that prevent changes from causing erosion.

Managing the Evolution

A Lattix project is an architectural representation of your application. It contains not just the map of all the elements and their dependencies but also the intended architecture, and the rules for the permitted dependencies. The project can also maintain an ordered worklist of the changes that you intend to make.

Since development is driven by external requirements, it never really stops. As a result, your architecture is also changing. Therefore, Lattix supports the notion of update. As your project changes, you can update it with the new source code allowing you to see what changed and whether those changes violated the architectural rules.

The Lattix project can also be set up in a web repository and the build system can be instrumented to automate the process of updating the repository. On every build you can see what the changes were, what the impact of those changes was on the rest of the system, and whether rules were violated. It is also possible to track key architectural metrics such as Stability, Cyclicality, and Coupling that measure the complexity of your system as it evolves.

Conclusion

Architecture is one of the most valuable artifacts to emerge from a legacy application. It contains the learning of how to tackle a complex problem. Even when a decision is made to end-of-life the legacy application, the architectural knowledge left over from the project will be vitally useful for the newer applications that will be created.

Try Lattix Architect on your Project

For more than 10 years, Lattix has helped refactor architecture, pinpoint problematic dependencies and manage the evolution of system design. Lattix provides support for a wide range of technologies including C/C++, Java, .NET, Fortran, Ada, Javascript, Actionscript, Pascal, Python, UML/SysML, Rhapsody, Sparx EA, Oracle, SQLServer, Sybase, LDI and Excel.

Do you want to see what your architecture looks like and what you can do to modularize it? We have helped companies all over the world improve the quality of software and we can help you achieve the same results.

Contact Lattix at sales@lattix.com or call 978-664-5050.

