NRC·CNRC

NRC Publications Archive
Archives des publications du CNRC

Toward upgrade risks assessment for OTS development

Putrycz, Erik

This publication could be one of several versions: author’s original, accepted manuscript or the publisher’s version. / La version de cette publication peut être l’une des suivantes : la version prépublication de l’auteur, la version acceptée du manuscrit ou la version de l’éditeur.

Publisher’s version / Version de l'éditeur:

Proceedings of the IOTSDM Workshop, ICCBSS, 2006

NRC Publications Archive Record / Notice des Archives des publications du CNRC :
https://nrc-publications.canada.ca/eng/view/object/?id=015a74c4-e126-446d-a5dd-069a8dc0e76
https://publications-cnrc.canada.ca/fra/voir/objet/?id=015a74c4-e126-446d-a5dd-069a8dc0e76d

Access and use of this website and the material on it are subject to the Terms and Conditions set forth at
https://nrc-publications.canada.ca/eng/copyright
READ THESE TERMS AND CONDITIONS CAREFULLY BEFORE USING THIS WEBSITE.

L’accès à ce site Web et l’utilisation de son contenu sont assujettis aux conditions présentées dans le site
https://publications-cnrc.canada.ca/fra/droits
LISEZ CES CONDITIONS ATTENTIVEMENT AVANT D’UTILISER CE SITE WEB.

Questions? Contact the NRC Publications Archive team at
PublicationsArchive-ArchivesPublications@nrc-cnrc.gc.ca. If you wish to email the authors directly, please see the first page of the publication for their contact information.

Vous avez des questions? Nous pouvons vous aider. Pour communiquer directement avec un auteur, consultez la première page de la revue dans laquelle son article a été publié afin de trouver ses coordonnées. Si vous n’arrivez pas à les repérer, communiquez avec nous à PublicationsArchive-ArchivesPublications@nrc-cnrc.gc.ca.

National Research Council Canada
Conseil national de recherches Canada
Canada

National Research Council Canada
Institute for Information Technology

Conseil national de recherches Canada
Institut de technologie de l'information

NRC · CNRC

*Toward upgrade risks assessment for OTS development**

Putrycz, E.
February 2006

* Proceedings of the IOTSDM Workshop, ICCBSS. Orlando, Florida, USA. February 2006. NRC 49330.

Copyright 2006 by
National Research Council of Canada

Permission is granted to quote short excerpts and to reproduce figures and tables from this report, provided that the source of such material is fully acknowledged.

Canada

Toward upgrade risks assessment for OTS development

Erik Putrycz
National Research Council of Canada
Software Engineering Group
erik.putrycz@nrc-cnrc.gc.ca

Abstract

Updates are a major part of the maintenance of every COTS-based application. Security flaws, bugs or missing functionalities can cause a vendor to reissue a new version and when major new functionality is added to the COTS component, vendors usually release a new major version. Every update - minor or major - is a potential risk for the existing functionality of the COTS-based application. This makes updating a difficult choice when a new version is available: will the new features and bug fixes be worth all the potential work to ensure existing functionality will not be broken? This paper presents a process and a risk scale factor that aims to evaluate the risk of updating of one COTS component in an application.

1. Introduction

A large part of the maintenance of COTS-based system is the upgrade process. NASA estimated in 2002, 400 candidate vendor patches per year, and more than 40 COTS upgrades per year. Many more patches have been necessary to address problems, especially that of security. Today applying upgrades to production systems requires a complex process to ensure the system operates correctly. At first, regression testing must verify that none of the existing functionality has been broken and that the upgrade integrates correctly with the other COTS products. Then the update must be planned and deployed on all the concerned systems.

This paper presents a process that aims to assess, before executing the update, the risk of breaking existing functionality by replacing one COTS component by a new version. The process proposed is focused on changes and impacts at the code level. Code level changes are usually the most expensive to implement. COTS-based applications consist of several COTS components and a glue code that connects all the components together. If another COTS component is not compatible, then the vendor is required to fix the issue. If the glue code connecting all the COTS component is affected by the update, the assembler (usually responsible of executing the update) can correct his/her code. Code level changes and connection can be obtained even with compiled programs using reverse engineering and simple disassembling with virtual machines such as Java or .Net.

At first (Section 2), the different types of connections between COTS components and their impact on updates and risks are discussed. Then, a process for assessing risk is presented in Section 3. Section 4 shows how this process is applied on a simple Java application. Finally, related works are detailed in Section 5.

2. Background

COTS applications are composed of several components and a glue code that the assembler uses to coordinate all the components together. The different types of connections of the components affect the consequences of the updates and the risks involved.

2.1. Types of connection

Components in a COTS-based application can be connected in many different ways. The connection types range from high level middleware communications to low-level runtime support (c.f. Table 1). In high-level types of connections, such as Web Services, the developers only need to describe the communications and low-level communications are delegated to external components (usually called middleware). Many middleware are even standardized by international organizations (w3c for Web Services and omg for CORBA), and help to ensure that communications take place correctly between all implementations. In this high-level type of connection, the coupling between the components is reduced to a simple interface that can be easily obtained.

In many other applications, especially legacy applications, the connection between components is made at a low level. It is common to see applications that communicate

using a low-level binary or text file between components (due to historical reasons). This implies that developers on every component respect the same file format and handle text parsing or binary details. This type of coupling increases the risk of the upgrade as bugs or new features may affect the file and generate errors in other components.

2.2. Types of updates

Two main types of updates are considered. minor version and major version. The details of each type of update are vendor dependant. In most cases, minor versions fix bugs, add new functionalities and don't break the specifications. Major versions may break previous APIs and other specifications. For instance, Service packs for windows known bugs and only minor new functionality (such as firewall for Windows XP) are added. Major releases (e.g. update from Windows 2000 to Windows XP) broke many existing APIs with newer versions of specifications (COM+, DirectX, etc.).

2.3. Types of risks

The risk caused by an upgrade depends on which element(s) of the COTS-based application is affected. If the glue code is affected, the risk is controllable since the same shareholder is in charge of maintaining this code and of upgrading the components.

If the upgrade affects another COTS component of the application, the risk is more important. In this case, it is possible that the upgrade breaks the functionality of the whole application. For instance, a change in the API of the component can prevent the application to execute correctly. In a Java application, a change of the API of a library will generate an exception at runtime that in many cases cause the execution to stop.

3. Assessing risk of updates

The objective of this paper is to propose a solution for assessing the risk of an upgrade by looking at the programming language level.

3.1. Objectives

The process proposed, aims to analyze the usage of one component by another without source code access using introspection, compiled code analysis and the analysis of other artifacts such as documentation and release notes.

In virtual machine based execution such as Java or .NET, the introspection capabilities enable API details to be queried without requiring source code access and byte-code disassembly can provide enough information to find dependencies.

By analyzing changes in the component being updated and the dependencies with the rest of the application, it is possible to calculate an estimate of the risk encountered.

3.2. Analysis process

The analysis process of the update risk consists of four steps:

1. Creation of a database containing all public artifacts for each COTS component;
2. Analysis of dependencies in the COTS-based system using code, release notes and documentation analysis;
3. Analysis of changes in the new version of the COTS component;
4. Assessment of the risk using the collected information.

Figure 1 presents an example of the process with two components A and B, where B is updated with a newer version.

At first, all the COTS components code are scanned using introspection or other techniques to build a list of all public artifacts (public types, functions, methods and other resources) that can be referenced by other COTS components. Then dependencies are collected using reverse engineering techniques and text analysis on the code plus documentation of each COTS component. After, changes in the new version of the COTS-component are discovered by comparing all the public artifacts of the new version with the identical artifacts of the previous version. The types of dependencies and the changes in the new version provide enough information to provide a report with a risk of upgrade. This scale is detailed in Section 3.5.

3.3. Types of changes in a component

In an object oriented system, the usage of one COTS component by another COTS component can take the following forms:

- class inheritance: inheritance of a class from another COTS component;
- field and variable declarations: fields and variables used in a class are of a type from another COTS component;
- method and function parameters: the parameters of one method (or function) reference classes from another COTS component;

Type of connection	Example of connection	Example of upgrade
Middleware communications	Web Service communications	Change in web service definition
Data centric-applications	Database for communicating between components	Database schema change
Programming Language level	A is used as a library of B	New major version of A
Runtime support	Component A is running on top of an operating system	New version of the operating system

Table 1. Types of Connection

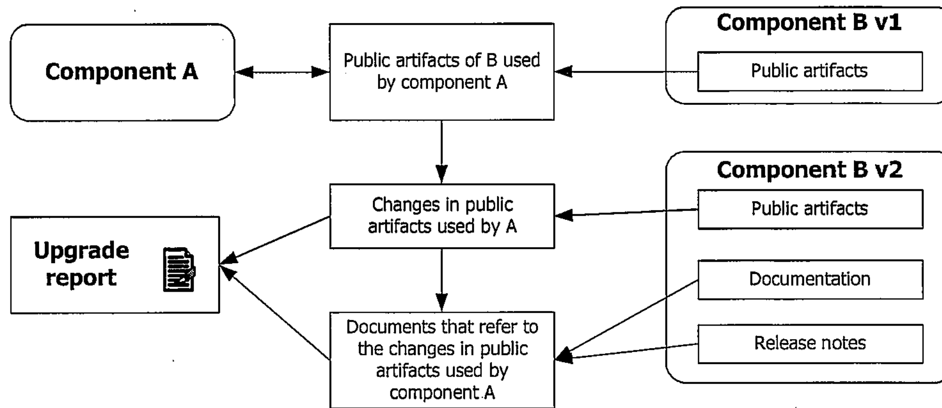

Figure 1. Example of the analysis process with two components

- static method or function call from another COTS component;
- meta information: modern languages such as C# or Java allow to add meta information descriptors from another COTS component to a class/method or function.

All the references from one component by another (fields, methods and inherited classes) are possible using the proper visibility level in the language. For instance in Java, only `public` classes and interfaces are visible by other components.

API changes between two versions of a component can be gathered by looking at the differences in the visible API. If possible, changes in the non public API can show if significant changes have been made internally.

When a source configuration management data such as CVS is available, changes in the source code can be taken into account and it is possible to determine precisely where the changes have been made. To calculate precisely the number of changes, non-significant changes (such as formatting or variable name changes) need to be ignored.

3.4. Analyzing dependencies

Once changes have been gathered between two versions of a component, it is necessary to know how the component is involved in the application and to find out how the rest of the application depends on the component being upgraded.

If source code is available, dependencies can be extracted by parsing the source code and tracking all references to external components. When no source code is available, compiled code for virtual machines (such as Java or .NET) can be analyzed and dependencies can be extracted. In Java, libraries such as Jakarta's BCEL or Objectweb's ASM [3] enable to disassemble compiled code and extract all type information. This way, it is possible to automatically build a map of dependencies between COTS components and the glue code. Also other dependencies can be extracted by analyzing the documentation; and searching type names, COTS product names and other keywords.

3.5. Risk factor scale

The risk scale proposed aims to capture:

- the break of dependencies causing the execution to fail;
- impacts on the connections used to communicate in the COTS-based application;

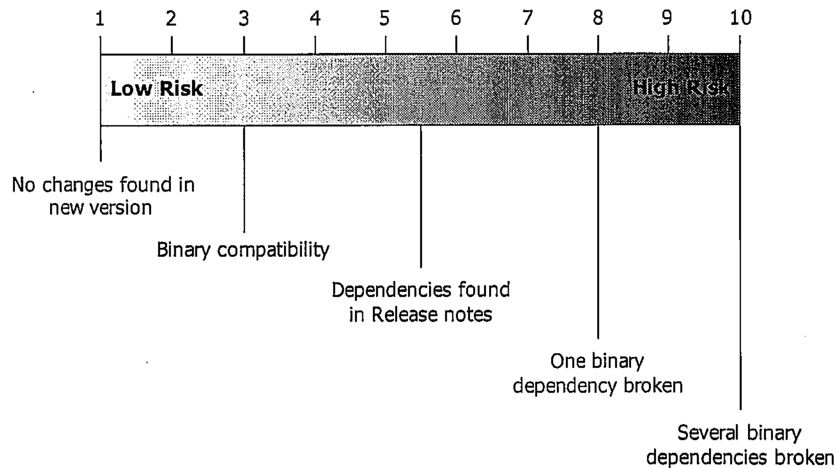

Figure 2. Risk Scale and risk examples for one updated component and one component affected

- amount of work required.

The break of dependencies happens when the public API that is used has been changed. This is considered the highest risk because it is very likely that the execution will fail. Other potential risks can be found when the interfaces used are mentioned in the release notes of the updated component. This usually signifies that internal changes have been made in the interface and causes a risk to introduce new bugs.

All these risks are ranked from 1 to 10 for each pair of components affected in the application. The final mark is the the maximum rank obtained between all pairs and depicts the risk for the whole application.

Ideally, the vendor or distributor of a COTS component could include an update report that lists the changes and give a risk factor on the impact of each one. This way, tools could make use of these risk factors and give more accurate estimates of the update risk.

4. Example

This section shows a simple example of how the upgrade risk can be analyzed. The example is a simple Java application with a single COTS component.

4.1. Java specific challenges

Method overriding was a major challenge when a subclass wants to override a method of the parent class, the class declares a method with the same name, same arguments and same visibility. Before Java 5, there wasn't any checking done that the similar method did exist in the parent class. So if the parent class is in a different COTS package and the parent method was renamed or removed, then the overridden method would never be called. In Java 5, Sun introduced the `@Override` annotation that is processed at compilation time and ensures that a similar method exists in the parent class. But Java doesn't implement any runtime mechanism to verify method overriding. As a consequence, when performing updates, overridden methods must be checked.

4.2. Application structure

The example application depends only of Log4J. Log4J[1] is a configurable and extensible logging framework for Java. The simple application is presented on Figure 3. The update considered is the update of Log4J version 1.2.4 to version 1.3-alpha7. Although the version 1.3-alpha7 doesn't have the final API, it already contains the major changes including class inheritance of `Category` and `Level` which are major elements of the framework. The simple application presented makes a non standard usage of `Category` and `Priority` in order to show consequences of the update. At line 11, the return type of `category.getChainedPriority()` is a `Priority` object. In Log4J 1.3-alpha7, the new return type is `Level` which is incompatible with the previous return type. The update will require to modify this code and replace `Priority` by `Level`.

4.3. Assessing the update risk

The first step of the analysis process consists in gathering all public artifacts for each COTS component and the glue

```
1      private static final Category category = Category.getInstance(SimpleProgram.class);
2
3      public static void main(String[] args) {
4          SimpleProgram program = new SimpleProgram();
5          program.init();
6          program.breakingCode();
7      }
8
9      private void breakingCode() {
10         category.error("I shouldn't use Category");
11         Priority willBreak = category.getChainedPriority();
12         category.error("error", new RuntimeException());
13     }
```

Figure 3. Example application

code. In this case, the Jar file (Java Archive file containing compiled classes) and the compiled example are being analyzed. All the Java types used are collected using Java's introspection capabilities.

Then, dependencies are extracted using the ASM library. All types (classes, interfaces and static methods) used in the simple example that belong to Log4J are collected. Dependencies are found two classes from Log4J: `Category` and `Priority`.

Next, the changes of Log4J are analyzed by comparing the new API with the existing one collected at the first stage. This showed that 148 public classes were removed and 30 public class were changed. Since `Category` was found in the changed classes, the release notes have been searched with the word `Category`. The search found the following sentence: *"...These changes are intended to enforce the rule that client code should never refer to the **Category** class directly, but use the Logger class instead."* This explains the cause of the incompatibility with the new version and this information will help the developer to correct the problem. The risk scale result in this scenario is 8 because of one significant binary incompatibility.

5. Related work

M. NorthCott looked in his Master's Thesis [8] at managing constraints and dependencies in a COTS system. He proposes a model for the software assembly process and for assembled software systems. In his model, a configuration consists of a graph of resources containing all the elements involved. This work is complementary and can allow a more complete representation of dependencies between components.

IBM released recently a tool called API Usage Scanner [6]. This tools is also based on ASM and scans Java JAR files to analyze the usage of an API. The usage of the API has to be described using a set of rules. The result of the execution generates a report that list all the usages described in the rules. The approach proposed in this paper doesn't require a developer to describe the API usage manually, the usage is extracted automatically by looking at changes between two versions of a component.

[9] presents a case study of the lessons learned during a massive nationwide, networked computer system upgrade of a government entity. This article discusses all the organization issues and is more focused on the deployment process as the work proposed aims to find risks before the deployment.

6. Conclusion

Updates of COTS components are a major element of the lifecycle of any COTS-based application. They are often necessary for security reasons or bug fixes but the amount of changes in the COTS component may introduce new bugs in the component which can break existing functionality.

This paper presents a process for assessing the risk of an update before deployment and aims to estimate the amount of changes in the new version of a COTS component that may affect the application. At the code level, this can be estimated automatically with reverse engineering, introspection and other techniques. Introspection allows the collection of information about public artifacts of all COTS components and disassembly enables the extraction of dependencies between the components involved. Introspection also enables the differences between the current version and the new version of the component (being updated) to be analyzed. Finally, a report is generated with all of the information related to the changes that impact the application, using release notes and documentation. In addition a simple risk scale captures the most significant factors in the update and gives a mark from 1 to 10 (where 10 is the most risky). This process has been applied with a simple Java application to explain how it can be used.

The research proposed in this paper is a preliminary proposal for assessing updates and the risk scale needs to be validated using several real examples. Also other factors such as amount of work required could be integrated in the

scale to bring more relevant results.

References

[1] Apache Software Fundation. Log4j. http://logging.apache.org/log4j, 2005.

[2] B. Boehm, W. Brown, and R. Turner. Spiral development of software-intensive systems of systems. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 706–707, New York, NY, USA, 2005. ACM Press.

[3] E. Bruneton, R. Lenglet, and T. Coupaye. "ASM: a code manipulation tool to implement adaptable systems". In *Proceedings of Adaptable and extensible component systems*, 2002.

[4] J. Dean, P. Oberndorf, M. Vigder, C. Abts, H. Erdogmus, N. Maiden, M. Looney, G. Heineman, and M. Guntersdorfer. Cots workshop: continuing collaborations for successful cots development. In *Proceedings of ICCBSS 2001*, volume 26, pages 61–73, New York, NY, USA, 2001. ACM Press.

[5] A. Egyed and R. Balzer. Unfriendly cots integration-instrumentation and interfaces for improved plugability. In *ASE '01: Proceedings of the 16th IEEE international conference on Automated software engineering*, page 223, Washington, DC, USA, 2001. IEEE Computer Society.

[6] A. Hui, A. Brodie, C. Sahoo, L. Nguyen, M.-R. Fisher, and R. Beasley. Api usage scanner: A java utility that scans java bytecode to detect references to targeted apis. http://www.alphaworks.ibm.com/tech/aus.

[7] A. F. Minkiewicz. Six steps to a successful cots implementation. *CrossTalk, The Journal of Defense Software Engineering*, 18(8), aug 2005.

[8] M. NorthCott. Managing dependencies and constraints in assembled software systems. Master's thesis, Ottawa-Carleton Institute for Computer Science, Carleton University, sep 2005.

[9] M. A. Raley and L. H. Etzkorn. Case study: lessons learned during a nationwide computer system upgrade. In *Proceedings of the 42nd annual Southeast regional conference*, 2004.