International Journal of Trend in Scientific Research and Development (IJTSRD)
International Open Access Journal | www.ijtsrd.com
ISSN No: 2456 - 6470 | Volume - 3 | Issue – 1 | Nov – Dec 2018

Tools for Software Re-Engineering

Prabhjot Singh Anand[1], Deepak Chahal[2], Latika Kharb[2]
[1]MCA Student, [2]Professor
Jagan Institute of Management Studies, Rohini, Delhi, India

ABSTRACT
Every day we use tools to help us achieve tasks; the use of a good tool in general will make a job much easier. When it comes to working with a computer, the tools we use are pieces of software which allow us to do our work. Without a text editor, it would be impossible to write a paper using a computer. In both the digital and real world, there are many options for tools for any job, and choosing the right tool can mean the difference between huge success and utter failure. For a software engineering project, the decision of what tools to use can make a large difference towards the cost and effectiveness of the project. Consider using a notepad text editor to write code for a project with tens of thousands of lines of code, it would be impractical at best.

This paper is designed to identify the major tools used in a reengineering project, and identify what choices are available for each of those pieces of software.

KEY WORDS: *Software re-engineering, UML, reverse engineering.*

1. Software Re-engineering

Software re-engineering at its core is the process of taking an old, possibly not working, application and making it new again. New software development is a well known process of taking requirements and turning them into code and a running application, but when re-engineering is considered, first the project team is reviving an existing asset to become the basis for this application.

Accordingly, the primary raw material for a reengineering project is the existing application. The first step in the process is to take the current version, and turn it back into the code which it came from. Two tools make this process possible: the hex editor, and the decompiler. Further, the code may be translated into a new language, and a design is then extracted from it. Debuggers, code translators and document generators aid greatly in this process. Finally, with the design created in UML, requirements can be derived, and the conceptual goals for the old piece of software are defined.

With the conceptual goals in hand, the changes needed to create a new product can begin. The forward-engineering half of the process mirrors a normal development schedule: requirements are derived from the goals, a design is created from the requirements, and finally code is written to match the design. With a reengineering project, however, a few extra pieces of information are in hand. The original design can be used to aid in creation of a new design, and the same for requirements and code. This allows for both a more streamlined process, but the tools will aid along the way. UML tools allow for the creation of good design, code generators create some of the code for us, IDEs and debuggers aid in the development process, and finally compilers and testing tools make the finished product work correctly.

For each of these tools, there are many options, and as was expressed, choosing the right tool matters.

2. Reverse Engineering - Implementation to Code

The first step in a software re-engineering project is to get some code to work with. The way that code is derived, when raw source code isn't readily available, is by decompiling the executable of the source application. Decompilers are used to increase the level of abstraction of code to a higher level: move from machine code to a human readable programming language. Since the machine code of a particular application was originally derived by compiling a specific language, a decompiler for that language is required.

2.1 Decompilers:

Some of the tools available in the market for decompilers are:

1. Jad Clipse (the JAD plugin for Eclipse)
2. Net Framework Decompiler
3. Mocha Decompiler

2.2 Comparison:

Decompilation is often a messy process, stripping code of comments, organization, or other helpful properties. This especially applies with certain computer languages: Intermediate languages like Java use formats that are closer to the original source than the native assembly codes you get from C++, which saves no metadata about the original code; decompiling a native language will give you less comprehensible source files that aren't as easily useful for reverse engineering. However, even in the best scenario for decompilation, you should use decompiled code on a case by case basis. Decompilers can be a shortcut to source code for analysis of a legacy application's design, but basing any new code directly off the decompiled code carries high risks. If the decompiled code is sufficiently obtuse, it may even be more efficient to analyze behaviors alone to determine the design and skip the source code step for parts of the project!

2.3 Feature Comparison:

For every computer language, a wide variety of decompilers are virtually guaranteed to exist. Operating with GUIs or via a command line, they provide translation from a lower level language to (hopefully) readable source code of varying degrees of quality, adding spacing and simplification so that programmers may interpret them. However, the quality of outputted code varies from decompiler to decompiler, and many errors can result depending on the process used by the chosen tool.

3. Code Translators:

Some of the tools available are:
DMS Software Reengineering Toolkit RES

3.1 Comparison

When to use code translators: The quality of code translated between languages will probably vary even more than the results of decompilers vary between each other. As such, thorough testing and behavioral comparisons with the legacy system are necessary when basing later, forward- engineered code off reverse engineered translations. Depending on the availability of quality translation tools and the relative size of the undertaking, it may be safer to have your programmers learn and discern design from the original legacy code.

3.2 Feature Comparison:

This description of code translators, in fact, is but a subset of the availability of automated code generation tools. Some are applicable in different phases than this one, such as tools that translate design (often in a UML-like format) into templates of connected code, or provide tools that generate interfaces in a desired language for access to a database system (such as the object-oriented database Objectivity). There are even tools that simply translate shorthand templates you write into full-fledged code. When leveraging these powerful automation tools, however, keep in mind that every automated step is one more level of abstraction between your systems' codebase and human-understandable code made by human programmers, as the forward engineering begins. If a team member codes part of a library, at least one person (said member) is privy to the reasoning behind its construction and how it works. If a portion of a library is auto-generated, there is no guarantee that anyone understands it, making it less maintainable.

4. Reverse Engineering - Code to Design Documentation Generator

The name is self explanatory. Documentation generators are extremely useful for reverse engineering. They save an enormous amount of human resources. Instead of manually going through the code and figuring out what the program does, documentation generators can do a great deal of it for you. They read the source program, based on the comments, the return types, parameters and definition of methods the document generators develop a small documentation with all this information. Though it is not complete, it is a good start. There are language specific generators and language independent generators. Some available ones are:

a) Java Doc
b) Php Documentor
c) Doc-O-Matic

4.1 Comparison

Documentation comes in a variety of formats, and the variety of generators reflects this. Some generators can produce diagrams and visuals of the interconnections between packages and classes, or embed hyperlinks between parts of the documentation in their class and method descriptions. Sandcastle, for example, is a project which generates MSDN-style linked and searchable documentation based on .NET source codes' comments and metadata. If the code of a legacy system to be reverse-engineered has enough metadata to warrant use of a documentation generator - or your programmers can record this metadata inline while inspecting the system's codebase - then look at the generators available for your current programming language, compare the metadata they require for effectiveness to the information available in your codebase, and then generate documentation to be expanded upon by your team as the legacy system's functionality becomes clearer.

5. Forward Engineering - Requirements to Design

As expected, the design phase in a re-engineering project mirrors the typical design phase of any software project. The requirements are used and verified to create a software design which fits. In this step, the most important tool is the Unified Modelling Language (UML) - more specifically the automated tools which allow UML to be effectively used.UML is a system of diagram styles and models which allow different essential parts of the design of a software system to be created. One primary example is the class diagram, which allows for classes in an object oriented design to be defined by their attributes and relations to other classes. UML tools support several major functional processes to make the design and coding step easier.

6. Diagramming

Diagramming is the most basic of tasks for UML, the creation and editing of the actual diagrams and models. The designers on a project will use diagramming to create the class diagrams, use case models, and many other models which represent the software design. This is the core functionality of a UML tool.

7. Code Generation

For a UML tool to have code generation features means that it can take the diagrams created and turn them into the basic code structure of the resulting program. In some versions, the user can provide a skeleton of the program source, in template form, and predefined tokens can be replaced with source code snippets as part of the code generation process.

8. Model and Diagram Interchange

XML Metadata Interchange (XMI) is the format for UML model interchange. XMI does not support UML Diagram Interchange, which allows you to import UML diagrams from one model to another.

9. Model Transformation

A key concept associated with the Model-driven architecture initiative is the capacity to transform a model into another model. For example, one might want to transform a platform-independent domain model into a Java platform-specific model for implementation. It is also possible to re-factor UML models to produce more concise and well-formed UML models. It is possible to generate UML models from other modeling notations, such as BPMN. The standard that supports this is called QVT for Queries/Views/Transformations. One example of an open- source QVT-solution is the ATL language built by INRIA.

10. IDE's

An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools and a debugger.

10.1 Features of IDE's

Visual Programming

Visual programming is a usage scenario in which an IDE is generally required. Visual IDEs allow users to create new applications by moving programming, building blocks, or code nodes to create flowcharts or structure diagrams that are then compiled or interpreted. These flowcharts often are based on the Unified Modelling Language.

10.2 Multiple Language support

Some IDEs support multiple languages, such as Eclipse, IntelliJ IDEA, My Eclipse or Net Beans, all based on Java, or Mono Develop, based on C#. Support for alternative languages is often provided by plugins, allowing them to be installed on the same IDE at the same time. For example, Eclipse and Netbeans have plugins for C/C++, Ada, GNAT (for example Ada GIDE), Perl, Python, Ruby, and PHP, among other languages in use.

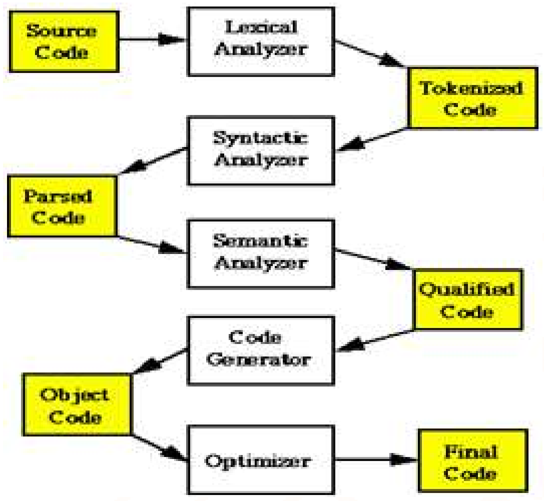
10.3 Multiple Platform Support
Attitudes across different computing platforms Support for various platforms like Linux, Macintosh, and Windows etc.

11. Debuggers
A debugger or debugging tool is a computer program that is used to test and debug other programs (the "target" program). The code to be examined might alternatively be running on an instruction set simulator (ISS), a technique that allows great power in its ability to halt when specific conditions are encountered but which will typically be somewhat slower than executing the code directly on the appropriate (or the same) processor. Some debuggers offer two modes of operation—full or partial simulation—to limit this impact.

11.1 Features of Debuggers
Typically, debuggers also offer more sophisticated functions such as running a program step by (a) step (single- stepping or program animation), (b) stopping (breaking) (pausing the program to examine the current state) at some event or specified instruction by means of a breakpoint, and (c) tracking the values of variables. Some debuggers have the ability to modify program state while it is running. It may also be possible to continue execution at a different location in the program to bypass a crash or logical error. Most mainstream debugging engines, such as GDB and DBX, provide console-based command line interfaces. Debugger front-ends are popular extensions to debugger engines that provide IDE integration, program animation, and visualization features.

12. Forward Engineering - Code to Executable to Production
12.1 Compilers
A compiler is a computer program (or set of programs) that transforms source code written in a programming language (the source language) into another computer language (the target language, often having a binary form known as object code). The most common reason for wanting to transform source code is to create an executable program.

Compilers are language Dependent. A few examples are C++ compilers, Java compilers, C# compilers etc.



13. Testing Tools
Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs. Thus to make the process of software testing more efficient and easy we use tools that help us to automate the process.

Conclusion
To sum up, tools can make or break your software. With the right tool, you can make wonders. The right tool makes your life easier but at the same time a wrong tool can be the end of your career! A tool cannot be selected just because it looks fancy or because it is easily available in the market. A tool has to be selected after all the considerations. A tool can change your life.

References:
1. http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell
2. http://www.andromeda.com/people/ddyer/java/decompiler-table.html
3. http://www.javaworld.com/javaworld/jw-07-1997/jw-07-decompilers.html?page=2
4. http://objectivity.com/pages/objectivity/c-net-programming-and-objectivitydb
5. http://www.infosys.tuwien.ac.at/Teaching/Courses/SWE/BellayGall-jsm98.pdf
6. http://e-archivo.uc3m.es/bitstream/10016/5697/1/knowledge_aler_KBS_2002_ps.pdf