



# SOFTWARE RE-ENGINEERING PROCESS

**Paul Briden**

**TESSELLA SUPPORT SERVICES PLC**

Issue V2.R1.M1

April 2000



## SOFTWARE RE-ENGINEERING PROCESS

### **Introduction**

This report provides a general overview of the software re-engineering process. Its main purpose being to give a brief high-level guide on how old, poorly documented, poorly structured code can be brought up to an acceptable and maintainable standard. It also touches on issues that could possibly be addressed during this process.

It is very much a generic overview and as such does not make reference to any specific platform or development tools.

### **The Scenario**

Generally poorly maintained and controlled code consists of a large number of program files spread over a number of locations or a single very large file with thousands of lines of code. There may appear to be a number of copies of the same file, all of which are subtly different. On viewing these files it is found that many have little or no comments and those comments that do exist are either meaningless or appear to bear no relationship to the code itself. Any associated documentation is out of date and refers to the initial version of the application that has undergone many undocumented modifications since then.

What does exist, though, is an executable file that is widely used to produce data upon which many decisions are made or actions taken.

### **The Decision**

A decision is then reached that some control over the application is required as it is about to be updated, ported or passed over to another person or department to maintain. This decision may well be related to the common situation where the only person in the company who knows anything about the application has handed in their notice or is approaching retirement. Also it is possible that the existing tools or platform it is being run on are about to be phased out due to obsolescence.

It is important that at this stage the actual objectives of any software re-engineering are identified. The cause of the decision may be obvious but there may be other underlying problems that have been around for years that now have the opportunity of being addressed. For example there may have been a problem with the existing application that users of the application have been consistently

working around (e.g. “It always produces a blank output page after every tenth sheet”, “It always takes 10 minutes to start up after I have run application A” or “Don’t press button B after doing that it always crashes”.) It may also present

the opportunity of adding in those extra features which have been waiting around for ages and everyone would really like to see implemented, or even to enhance or add in any security elements.

### **The Process**

A re-engineering process, which can be used to take control of the situation, can be divided into 11 basic steps. These are:

1. Identification of Existing Source Code/Documentation
2. Verification of Identified Code and Setting up of a Baseline
3. Familiarisation with the Existing Application
4. Identification of Re-engineering Requirements
5. Writing a Re-engineering Plan
6. Production of a Test Plan, Test Data and Script
7. Producing Test Results for Original Version
8. Producing Test Results for Original Ported Version (unmodified) - if applicable
9. Restructuring the Code (in identifiable phases)
10. Testing the Code (after each phase)
11. Updating/Writing Documentation

It cannot be over emphasised that time spent on the initial stages can save a lot of effort later on. Cutting back on time in these stages can lead to major costs later on if it is found that you have been re-engineering the wrong code!

These stages will now be described in turn.

#### **1. Identification of Existing Source Code/Documentation**

The first stage of the re-engineering process is the identification of the program files that are the basis for the executable being re-engineered. The developer who is to re-engineer the application needs to identify the source files and this is best achieved in close consultation with the appropriate person in the company whose application is being re-engineered. Generally there is a person within the company who knows intimately all of the files required and can bring some logic to their existence.

Also during this stage any existing documentation should be gathered and copies of any relevant third party software required by the application identified and obtained.

All this information and the identified files should be placed under a new controlled structure, which will be defined in a Configuration Plan. This plan identifies all the required components, their version numbers (if applicable) and where they reside. It will include the currently used executable.

During all stages of the re-engineering process the developer should keep a log of all pertinent facts and where they originated (e.g. this file was selected because X said that it included the revised value for parameter Y. X said that he used the compiler version 1.4 to create the executable.)

## **2. Verification of Identified Code and Setting up of a Baseline**

Hopefully all of the required files to build the application can be identified. During this next stage the application is built to recreate the existing executable. It may require some investigation of the compilation options used to exactly recreate it and it is possible that when the existing executable was built no record was kept of these. A lot of investigative work may be required at this stage and many things can affect the created executable (e.g. the versions of third party products used, the version of the compiler, compiler options etc.)

Once what is thought to be an exact copy of the existing executable is created it should be tested against this on the same platform to verify it really is the same. This can be achieved by using sample input data or test files. Any such test files or data that is retained for future reference or testing along with the output produced and which version of the application produced this output.

The source code files and any associated data files required to run the application should be documented and will form the baseline for the existing version. A copy of all these files must be kept and identified in a Baseline Description along with the version of any third party products which may be required and of any supporting documentation or development tools (i.e. compiler versions).

## **3. Familiarisation with the Existing Application**

It is now important for the developer, who is to re-engineer the code, to become familiar with it. He needs to know what it is basically trying to do and how it goes about this. This can be gained by trying the application out under a variety

of situations, carrying out code reviews/walkthroughs and discussions with its users. Access to someone familiar with the code is essential at this stage.

During this phase the developer can start introducing comments into the existing code which will help when coming to restructure the code. Other observations can be placed in his project log, such as ideas of improving the code, descriptions of any relevant algorithms used, a flow diagram etc..

It is not uncommon that during this stage bugs are found. How these are dealt with needs to be discussed with the customer. Generally, any bugs identified will be noted at this stage and given a unique identifier and returned to during the re-coding of the application.

#### **4. Identification of Re-engineering Requirements**

The next stage is to identify exactly what is required from the re-engineering process. This could be one or more of the following:

- Comment the code
- Modularise the code
- Document the code
- Enhance the code
- Identification and removal of unused code
- Identification and removal of duplicated code
- Identification and removal of duplicated parameter/data definitions
- Port the code to another platform
- Enhance the User Interface/Bring it up to date
- Bring the code up to a defined language standard
- Convert the Code into an alternative development language
- Improve/Maintain the performance
- Optimise the code
- Improve internal error handling
- Include additional functionality
- Fix existing bugs
- Use alternate third party products

Each of these can be broken down into a set of software requirements defining

what the re-engineered code must do.

### **5. Write a Re-engineering Plan**

Once the re-engineering requirements have been identified a plan needs to be written on how these are to be implemented. To maintain control over this process it should be broken down into distinct phases, examples of which are listed in the previous section. At the end of each phase the code should be baselined, that is a copy of that code at that point in time kept. This code will then be used to recreate the executable to verify that it still agrees with the original version. Carrying this out in a phased manner means that any problems introduced during the re-engineering process can be quickly identified and the cause eliminated or addressed.

This plan will need to be agreed with the customer before proceeding onto the next stage.

### **6. Production of a Test Plan, Test Data and Script**

To enable the application to be tested and verified a test plan and script need to be written. The test plan essentially explains how testing will be carried out and the test script provides a prescriptive list of actions to carry out to perform the test. The test script consists of a list of test cases which the developer (or possibly an alternative person brought in to carry out the testing) follows and ensures that the same test can be carried out consistently each time. Generally each test case will consist of a unique identifier, a brief description of the procedure required to carry out the test, the expected result of carrying out those actions and an area where the tester can identify whether the test was successful. The test script should also allow the tester to identify the version of the application being tested, the platform (and ideally the actual machine) it was tested on, the version number of the operating system and any associated third party products used, the testers name, the current date and any other pertinent information.

To assist with the testing a number of input data files will probably be required and a set of output files will be created. These output files should be stored and used to confirm agreement with the original version. By this stage any required changes to the input/output file formats need to be confirmed.

### **7. Producing Test Results for Original Version**

The test script, using the defined input data, should be run for the original version of the application. This will produce output data or data files which will be used

to verify the re-engineered code.

At this stage the performance of the code may be measured if this is an issue.

### **8. Producing Test Results for Original Ported Version (unmodified) - If applicable**

If the code is to be ported onto a different platform then it is imperative that an unmodified version is initially ported to the new platform and run using the test script. It may not be possible to port it without making some required changes. If any changes are required these should be as few as possible and documented. Examples of such changes are having to replace the use of platform specific language extensions which may have been used.

The ported code should then be run through the same test script using the same test data. It must be appreciated by the customer that porting code from one platform to another will not necessarily lead to exactly the same output results being produced. Hopefully the output should agree with the original version but this certainly may not be the case.

Different platforms/operating systems handle numeric data differently and different compilers work in different ways. This may lead to small rounding error differences in the output when the application is run on the two platforms. In other applications, especially modelling applications, it is possible that in certain situations very large differences may be seen. This is generally caused by the model making use of small differences between large numbers and is a limitation of the model as represented in the application. Consequently, if large differences in the output are seen, this does not necessarily mean that the code being run on either platform is wrong but may indicate that the task being tackled has not been well represented within the code or at worse the underlying model/algorithm is not suitable for computational representation. It is also possible that the differences seen could be traced to differences in compiler options on the two platforms or related to errors introduced when modifying any platform specific code during the porting exercise.

When differences are seen between the original and ported versions they need to be discussed with the customer and a course of action agreed. This may simply be to note the differences and proceed with the re-engineering.

If performance is an issue then this should be measured now and compared to

that on the original platform. This may lead to the identification of issues connected with running on the new platform and whether the specification of the new platform needs to be improved.

Once all issues have been resolved the output produced on the ported original version will form the basis against which all re-engineered code will be compared and not the original version on the original platform.

## **9. Restructuring the Code (in identifiable phases)**

The original code should now be re-engineered according to the phases identified.

During the restructuring, software tools may be used to help identify or investigate the code coverage, hot spots, unused variables, checking the code conforms to a coding standard etc. Manual code inspections can also be used to help identify and then remove duplicated code, optimise array storage, collate parameter definitions etc.

The restructuring of the code is an ideal opportunity to improve its performance. There are many not so obvious issues that can significantly affect the performance of written code. An example is in the storing and accessing of data in arrays. Different languages store array data in a computer's memory in different ways. The way this data is accessed can have a drastic effect on the applications performance in some cases. This is something very well known to the seasoned developer but may not have been appreciated by the original code writer. Other performance improvements can be obtained if a more recent version of the programming language is being used during the re-engineering as this may offer faster ways of handling certain aspects of the original design.

## **10. Testing the Code (after each phase)**

After each phase is complete the system test script should be run on the re-engineered code and the output compared to that produced by the original code. Only after agreement is reached should the next phase be started.

After each stage the code should be baselined and an identifiable copy kept along with the test associated results. The storing of baseline copies of the code will allow the developer to return easily to an exact copy of the code as it existed at the end of any of the defined phases. This is very useful for tracking bugs that may appear later and to identify at what point they were introduced. It also

offers the customer the opportunity to formally release a copy of the code at the end of any of the defined phases. Such a release may not always be planned in advance and it is possible that there may be a sudden requirement to have an intermediate release as soon as possible (or even instantly). In these cases the developer can simply return to the last baselined version and be confident it will work and thereby keep the customer very happy with a very fast response to their request.

## **11. Update/Write Documentation**

Finally once the re-engineering process has been completed and tested any existing system documentation should be updated or, if none exists, written. Documentation is a very important part of the re-engineering process as it, along with the comments within the code, is the primary source of information that will assist in the future support and maintenance of the application. A typical customer requirement put forward when re-engineering existing code is the desire to move away from the dependence on a single person for its support and future development. Adequate documentation and commented code should mean that any developer familiar with the appropriate programming language can quickly get up to speed with the code and provide a high level of support in a short period of time.

The content of the documentation is very dependent on the type of application that has been re-engineered and of other supporting documentation that may already exist. One area, though, that should be covered is a description of any fundamental changes that were introduced during the re-engineering process. It should also refer to or include the details of the configuration plan and the defined baselines that were created during the re-engineering process. Other areas which should be documented, if appropriate, include:

- Description on how to install/uninstall the application and how to verify which version is installed.
- A quick guide which describes how to use the application.
- An overview of the application design. (This can be more detailed if required).
- Details on how to build the application ready for release.

## **The End**

The final re-engineered code should be baselined along with the associated documentation and the project log.

The re-engineered application is now ready for release and is in a well-defined state which will make any future support and maintenance much less arduous.

**Tessella Support Services plc**  
**Creating Software for Science and Engineering**

Tessella's services range from feasibility studies, through system design, development, implementation and ongoing support. Our expertise includes:

Data Analysis Software  
Data Capture  
Simulation Software  
Advanced Graphics  
Systems Support  
Database Applications

**Other Technical Supplements available include:**

- |   |  |
|---|--|
| <input type="checkbox"/> Archiving of Electronic Info   | <input type="checkbox"/> Object Oriented Programming   |
| <input type="checkbox"/> Active Server Pages            | <input type="checkbox"/> Pocket PC                     |
| <input type="checkbox"/> Automated GUI Testing          | <input type="checkbox"/> Portable GUI Development      |
| <input type="checkbox"/> Bayesian Statistics            | <input type="checkbox"/> Printer Technology Guide      |
| <input type="checkbox"/> Beowulf Clusters               | <input type="checkbox"/> Real Time Systems             |
| <input type="checkbox"/> C++                            | <input type="checkbox"/> Regression Testing            |
| <input type="checkbox"/> Client-Server Technology       | <input type="checkbox"/> Security and the Internet     |
| <input type="checkbox"/> COM                            | <input type="checkbox"/> Simulation                    |
| <input type="checkbox"/> Computational Fluid Dynamics   | <input type="checkbox"/> Soft Computing                |
| <input type="checkbox"/> Computer Image Processing      | <input type="checkbox"/> Software Design Methodologies |
| <input type="checkbox"/> Decision Support Systems       | <input type="checkbox"/> Software Development Cycle    |
| <input type="checkbox"/> Electronic Data Capture        | <input type="checkbox"/> Software Documentation        |
| <input type="checkbox"/> Electronic Lab Notebooks       | <input type="checkbox"/> Software Portability          |
| <input type="checkbox"/> Excel                          | <input type="checkbox"/> Software Re-engineering       |
| <input type="checkbox"/> Extending the Life of Software | <input type="checkbox"/> Software Specification        |
| <input type="checkbox"/> Federal Drug Administration    | <input type="checkbox"/> SQL                           |
| <input type="checkbox"/> FORTRAN 90                     | <input type="checkbox"/> UNIX Inter-Process Comms      |
| <input type="checkbox"/> Grid Computing                 | <input type="checkbox"/> UNIX Systems Performance      |
| <input type="checkbox"/> High Throughput Screening      | <input type="checkbox"/> UNIX Workstations             |
| <input type="checkbox"/> Instrumentation                | <input type="checkbox"/> Visual Basic 6                |
| <input type="checkbox"/> Integrated Lab Systems         | <input type="checkbox"/> WAP                           |
| <input type="checkbox"/> J2EE                           | <input type="checkbox"/> Web Services                  |
| <input type="checkbox"/> Java                           | <input type="checkbox"/> Windows 2000 Services         |
| <input type="checkbox"/> Lims                           | <input type="checkbox"/> XML                           |
| <input type="checkbox"/> Linux                          | <input type="checkbox"/> X Windows                     |
| <input type="checkbox"/> Microsoft Net                  |  |



INVESTOR IN PEOPLE

**Tessella Support Services plc**  
3 Vineyard Chambers, Abingdon, Oxon, OX14 3PX, England  
Tel: (+44) (0) 1235 555511 Fax: (+44) (0) 1235 553301  
E-mail: [info@tessella.com](mailto:info@tessella.com) Web Address: <http://www.tessella.com>