Reverse engineering to ensure re-use of an existing well established railway simulator

W.S. Chan, S. Shao, J. Allan

London Underground Ltd. Professional Services Group,
Telstar House, Eastbourne Terrace, London W2 6LG, UK

INTRODUCTION

Background.
The Multi-Train Simulator (MTS) is capable of modelling all the major sub-systems including Power Supply, Rolling Stock and Signalling & Control Systems on rapid transit railway systems. It is important to stress that interactions between sub-systems have a significant effect on the performance of the whole system. This feature was built-in implicitly in the original design of the MTS and developed in the Traction Research Group at the University of Birmingham more than 20 years ago with an investment in excess of 30 person years [1,2].

MTS and Its Applications.
The MTS has proved to be a useful and valuable tool in quantifying engineering problems in the design stage of rapid transit railway projects. It has been used both in the UK and worldwide by traction manufacturers and railway operators in assessing the whole Electrical & Mechanical (E&M) system, as well as individual sub-systems such as Power Supply, Rolling Stock and Signal & Control Systems. Typical applications carried out include:-

a) Whole System studies on energy consumption for a given time-tabled operation. This includes energy reduction and saving, resulting from the use of regenerative braking.

b) Power System design studies to quantify substation capacity, feeder cable breaker and conductor rail rating.

c) Train Performance design studies to assess the adequacy as well as relative merits of traction equipment to meet the required performance.
d) Signal System studies to investigate signal behaviour at minimum headway operation with a view to optimise train movement in both automatic and junction areas.

Other features and applications of the MTS are listed in a number of published papers [2, 3, 4, 5].

The Need For Reverse Engineering.
London Underground Limited purchased this well-established MTS from the University of Birmingham in 1989. Apart from using the package as the main tool in power system designs on major engineering projects such as the Central Line Modernisation Project and the Jubilee Line Extension Project to meet the immediate needs, a multi-stage development programme has been initiated to further develop the MTS to become an Integrated Railway Simulation Package encompassing all railway aspects. A joint development programme was set up with Hong Kong Mass Transit Railway Corporation (HKMTRC) [4]. The aim of the programme was to establish a standard base software suite by working collaboratively on simulator development adopting the same standard, methodology and CASE tools. Hence, new modules applicable to both LUL and HKMTRCs needs or application specific modules to suit the special needs of either operator can be added using the same standard.

It is widely recognised that a significant proportion of time and cost will be allocated in maintaining the software throughout the software life cycle. In addition, it costs more to maintain poorly and insufficiently documented software. It might also lead to incorrect identification of the right location to carry out modifications. This task will be even more difficult for a complex program, like the MTS, with 16,000 command lines. The adoption of a structured method will enable software development to be carried out more cost-effectively as problems can be identified in the early stage of the development. To fulfill LUL's commitments to quality and safety, two options were considered: the option of completely re-writing the existing simulator in a new environment using modern standards was not justified on the basis of significant time and resources involved. It was decided instead to re-structure the code and bring the software documentation and development, up to modern standards before carrying out further development. Thus, the process of reverse engineering was carried out before any forward engineering was performed in a structured and cost-effective way.
REVERSE ENGINEERING PROCESS

**Choice of Methodology and CASE Tools.**

Before choosing the methodology and CASE tools, a brief review of existing methodologies together with currently available CASE tools was carried out in order to determine their suitability for this type of application. Some representative methodologies considered include:-

a) Structured Systems Analysis and Design Method (SSADM)/LSDM.
b) Jackson Structured Design (JSD).
c) Yourdon Structured Design Methodology.

It was found that LSDM/SSADM had been widely used by the UK Government and other organisations in developing non-technical information based projects. The LSDM methodology has also been adopted for business applications by London Underground Limited.

After studying the above well known software methodologies, it was found that the majority of them stress the importance of data analysis by describing data structures in entity relationship diagrams which is particularly appropriate for information based business software. On the other hand, the Yourdon Structured Design Methodology emphasises the importance of processes, treating systems as processes linked by data flow which is more suitable to describe process driven simulation system and also appears to be widely used in the technical software industry.

The choice of the CASE tool was recommended by an external consultant based on the following criteria specified:-

a) Suitability for technical software applications.
b) Suitability for reverse engineering.
c) Full support for the required diagrams and descriptions.
d) Single repository for all design information and data which enables flexible reporting.
e) Automatic consistency checking facilities.
f) Potential for expansion on a network environment but initial use possible on a P.C.

The consultant used the STARTS GUIDE as the basis to search for a suitable CASE tool from the available products on the market.
Following a non-exhaustive analysis, use of the Excelerator CASE tool was recommended for this project. The Yourdon Structured Design Methodology was adopted as the basis of the work with the use of Excelerator CASE tool to support the methodology in defining and producing system development documentation. The final code remained in Fortran 77.

**Approach Adopted.**

In view of the complex nature of the project and lack of internal resource at that time, the decision was made to appoint an external consultant to carry out the initial part of this Reverse Engineering Project. As this was the first Reverse Engineering Project undertaken by LUL, careful steps had to be taken. It was decided to start the reverse engineering work on a relatively small portion of the whole simulator by using a well established consultant with previous experience in reverse engineering technical software projects. Throughout the project, the work by the Consultant was closely monitored by LUL’s engineers who have indepth knowledge of the simulator. At the end of this initial work, LUL has gained sufficient confidence in this project to use a well qualified Contract Engineer, with extensive railway and software engineering training/experience, to carry out the remaining reverse engineering work under LUL supervision.

Prior to the start of the main reverse engineering the data structure of the software was carefully rationalised to aid readability and to improve modular functionality. The opportunity was taken to structure the code in a way suitable to adopt an object oriented approach in the future. Finally all Fortran code was updated to Fortran 77. Then with the code fixed, the key issue was to ensure the documentation produced matched the code. The approach adopted was first to analyse the source code in order to re-construct the algorithms used in the modelling process. This was followed by producing a ‘first-cut’ functional specification and program specification. A detailed system walk through was then carried out to check the coherence between the program specification and the program code. With the program specification in place, the functional specification was then reviewed to ensure coherence between the program specification and the functional specification. A test specification was also derived from the user requirements specification simultaneously. A detailed breakdown of the above processes is shown in Figure 1.
Documentation Required.
The Yourdon methodology was tailored to our requirements.

Additional emphasises on the details of documentation were:

a) The philosophy behind the design of the system was included. For example, it contains not only the description of the program structure, but also the reason why such a structure has been chosen.

This was extended down to code level. It does not simply describe how the code works and what the code is doing, but also includes the reasoning behind the design.

b) The paper documentation is concise. Unnecessary quantity and diversity of paper work has been avoided. Data stored in the Excelerator database can be presented in different formats. Only relevant parts of the data are included in the paper documentation with an explanation on what other information is available for different applications.

The next section describes the basic frame work of the development documentation defined at the beginning of the work. The emphasis was on the adequacy of graphical representation of data and process flow to aid understanding of the system.

a) **Functional Specification.** This provides a complete overview of the system and describes the functions provided by the system. It contains:

i) Context diagram showing system environment.
ii) Data flow diagrams (DFD) of the main system showing the relationship between all the subsystems.
iii) Subsystem DFDs.
iv) Description of each subsystem as identified on the DFDs.
v) Input descriptions of the system.
vi) Output descriptions of the system.
vii) Process descriptions.
viii) Data model diagrams.
Other textual and graphical information includes:-

ix) Entity description.
x) List of contents in each entity.
xi) Data element descriptions.

b) **Program Specification.** This contains a detailed description of program structure and each module. It includes:

i) Physical structure diagram
ii) Logical program structure.
iii) Data structure diagrams.
iv) High level structured English description for each module.
v) Input/Output lists of each module.
vi) Structured Decision Table.

c) **Test Specification.** This defines the tests to be performed on the software in order to validate its performance against the requirements specification, integration testing and module testing. It includes:-

i) Extract of requirements.
ii) Implementation of requirements.
iii) Test strategy.
iv) Test description.

Other documentation specified is shown in Figure 2. Further tabular information for re-engineering in the future is also stored in the Excelerator database.

**Work Undertaken.**

After analysing the code, a structured chart was formed to illustrate the functional hierarchy of the system. Each box in the chart represents a program module (a subroutine in Fortran language) (Fig.3). The arrows represent the data flow between modules. A list of elements in each direction of the data flow was also derived and recorded in the data dictionary simultaneously.

The next step was to define the system boundary and the interfaces between the system and external entities. A context diagram as shown in Figure 4 was constructed for this purpose. The bubble
Software Quality Management 587

represents the system process while each box represents an external entity such as input data required and output data generated.

All the global variables used in the MTS are contained in an include file. Therefore, the next task was to partition all this data into seven external entities and two internal entities. This was then followed by constructing the top level data flow diagram (Fig. 5). All other subsystem data flow diagrams were then constructed. From the data flow, all the elements are grouped into relevant records of the data flow recorded in the data dictionary. Hence the data model diagrams can be established. Figure 6 shows the track data model diagram which describes the structure of the track data. The challenge to the engineer in the above process was to group all the elements into correct stores, as well as avoiding duplication and minimising similarity of data records.

Although the CASE tool was able to check the consistency of all the data flows, significant limitations on the tool were found and are discussed in later sections.

In the previous processes described, two major components which constitute the essential model of the Yourdon's Methodology have been constructed. These two models are:-

a) ENVIRONMENTAL MODEL - it consists of a context diagram, description of the purpose of the system, subsystems and subroutines.

b) BEHAVIOUR MODEL - it consists of data flow diagrams, data dictionary, data model and process description.

It is important to include the use of design structure diagrams to illustrate the process flow and the logical sequence of each subroutine. This was represented by the adoption of Jackson's structure diagram which is similar to the diagram specified in British Standard BS 6224. An example of this diagram is shown in Figure 7.

The next level down into each subroutine is the textual description of the code using the Structured English Description, which is also termed pseudo-code. In the case of a subroutine containing a significant amount of complex logical statements and decisions, the use of decision tables have proved to be more effective to describe the logical conditions considered and the corresponding actions required. They can also be
used to check missing action and associated logical error. An example of these two methods are given in Figures 8a and 8b.

The derivation of the test specification started from the user requirements specification as each requirement is linked directly to one or more processes which implement this requirement. It includes module testing and an integration test. The tests were then derived to check the accuracy of each process in satisfying the requirement. In some cases, a test case has been derived to satisfy a number of processes. Detailed discussions had been held to decide the range and depth of tests required to sufficiently test the program within the financial constraints.

EXPERIENCE LEARNT

The reverse engineering project was carried out using the Yourdon Methodology as the basis with local customisation on the documentation required. It is believed that the methodology can be further optimised and refined to suit different applications. During the process, it was found that a rigid bottom-up approach was not appropriate. It actually involved constant cross-checking between different levels of documentation using both human intelligence and various CASE tools. The success of the project relied on a system analyst with indepth knowledge of the methodology and the CASE tool, as well as an engineer who had indepth knowledge of the program and the philosophy behind the models.

A clear audit path can now be provided on the program. As a result of this process, some drawbacks of the system have been identified, modifications are now being carried out in the forward engineering path to overcome system limitations.

Drawbacks of Existing CASE Tools.

a) At the time of starting this work, three years ago, a complete CASE tool to support both the design and coding stage work was not available, not to mention a tool which could provide the whole life cycle support, as well as facilities suitable for reverse engineering. Therefore, checking of coherence between the code and documentation was carried out manually using a combination of an off-shelf code analysing package and a Fortran Program analysing tool developed in-house. This verification process can
be performed more cost-effectively and efficiently if the CASE industry could provide an integrated tool at least to bridge this important gap.

b) Although the Excelerator CASE tool is capable of supporting a multi-user environment, it is not suitable to support development carried out simultaneously by two organisations 10,000km apart. As the graphical data cannot be ported partially from one system to the other, it means that modifications carried out on the same graph simultaneously by both organisations need to be entered manually again.

c) As there is no established standard in the CASE industry, it is virtually impossible to port all the information from the repository of one CASE tool to that of another CASE tool. It means that a project needs to stay in the same CASE tool throughout its life or all the information has to be entered again if the decision was made to move to another CASE tool. For example, we have moved our development platform from a P.C. to a SUN workstation but all the information is still stored on the P.C. version of the CASE tool.

d) In representing the hierarchy of the whole system using the structured charts, it was found that even with a modest size program like the MTS with 112 subroutines, three separate charts were required to represent the whole system due to the limited memory space to put all of them in one chart. Even with this approach, some arrows were found missing from the charts.

e) In constructing the data flow diagrams (DFD) it was found that for a complex DFD

(i) some arrows would not touch the bubbles
(ii) the avoidance of cross over between arrows had proved to be a difficult task to achieve.

CONCLUSIONS

a) This paper has illustrated the approach adopted to reverse engineering, an existing and well established railway simulator. It has also been demonstrated that it is possible to jointly develop a
complex simulator by two different organisations by adopting the same standard, methodology and CASE tools.

b) Following the successful completion of this project, forward engineering has now been applied to continue further development of the project.

c) Further reverse engineering projects are currently being considered on software developed previously in a University research environment.

d) The CASE industry has still yet to set up an agreeable common standard to enable data transfer and provide an integrated tool which can truly support and automate the software engineering throughout the software life cycle.

REFERENCES


ACKNOWLEDGEMENT

The authors wish to thank the Engineering Director of London Underground Limited for permission to publish this paper.
FIG. 1
Reverse Engineering Process

EXISTING SOFTWARE

DATA MODEL DIAGRAMS

DATA DICTIONARY

DOCUMENTATION PRODUCTION

SUBSYSTEM DESCRIPTION

FUNCTIONAL DESCRIPTION

COMMUNITY

structured english description

TEST PLAN

USER DOCUMENTATION

SOFTWARE CODING DESIGN

PROGRAM SPECIFICATION

POWPLAN

FUNCTIONAL SPECIFICATION

PROGRAM DESIGN

TEST SPECIFICATION

TEST DESIGN

TEST REPORT

User Requirement

Test Reports

CLIENT BRIEF

FUNCTIONAL DESIGN

SOFTWARE CODING DESIGN

CLIENT/USER REQUIREMENTS SPECIFICATION

PROGRAM DESIGN

CODING REPORT

FORWARD ENGINEERING (TOP-DOWN DESIGN)

REVERSE ENGINEERING / RE-ENGINEERING

FIGURE 2 DOCUMENTATION PRODUCED IN SOFTWARE DESIGN
FIGURE 3 MTS STRUCTURE CHART

FIGURE 4 MTS CONTEXT DIAGRAM
FIGURE 5 FIRST LEVEL DATA FLOW DIAGRAM

FIGURE 6 DATA MODEL DIAGRAM OF TRACK
FIGURE 7 STRUCTURE DIAGRAM OF 'SIMULATE'

2.1.2 findline
   do ( for each 1st substation
   do ( if substation connected on 2-road
      then do ( for each 2nd substation
         do ( define the 1st line no.)
            ( see decision table dt-2.1.2 )
         do ( for each substation
            do ( define the final line no.)
               ( see decision table dt-2.1.2 )
   FIGURE 8a AN EXAMPLE OF STRUCTURED ENGLISH DESCRIPTION

   dt-2.1.2 c 2nd sub type r r r r 1 1 1 c c c c c c
   c 1st sub type l l c c R R c c C c c c c
   c routes from 2 2 4 4 2 2 3 3 2 2 4 4 4 4 3 3 3 3 3 3
   to 2 4 2 4 2 4 2 4 2 5 2 6 5 6
   ***********************************************
   a first line no. 1 3 1 3 1 3 1 3 1 3 1 3 3 3 3 3 3 3
   a a error y
   ***********************************************
   For each substation:
   c sub type r c c L R c
   **************************************************
   a final line no. > 4 y y y
   a final line no. = 1st line no. + 1 y y y

FIGURE 8b AN EXAMPLE OF STRUCTURED DECISION TABLE