Reengineering Software – A Case Study

Megan Graham & Michael Oudshoorn
Department of Computer Science
Montana State University
Introduction

• Legacy Software
  – Introduce the Presenter program as example legacy software.

• Applying Reengineering Practices
  – Design concepts, tools, and architectures as applied to the Presenter program.

• Metrics
  – Measuring the success of reengineering.
  – Results from measurements.

• Conclusions and Future Work
Legacy Software

• Legacy software is critical software that can not be modified efficiently.
  – Often it is software developed in-house which remains valuable to the organization.
  – Outlived and outgrown original design purpose.
    • Original development methods outdated.
    • Original developer(s) often not available for consultation.
  – Monolithic
    • Patches and modifications
    • Poor documentation
  – High maintenance costs...
• Organizations tend not to consider reengineering until the maintenance costs exceeds some perceived acceptable level. This level of effort is ill-defined and poorly understood.
Maintenance Effort Distribution

- Fault Repair (17%)
- Software adaptation (18%)
- Functionality addition or modification (65%)
Maintenance Costs

- Consider reengineering if a significant amount of resources are devoted to adding or modifying functionality.
- Maintenance costs increase with the age of the software – maintenance becomes more difficult.
- Research has shown 50%-80% of development time is spent on maintenance. It may account for >90% of the total development costs.
- ~75% of software maintenance costs relate to adaptive and perfective maintenance.
- If a system is relatively static and near the end of its useful life, then maintenance is likely to be more cost effective than reengineering.
Software Maintenance Costs

- Is it worth the cost of reengineering to decrease maintenance cost?
- Newly developed code costs estimated at $6-$26 per line of code. Reuse of code estimated to save 80% of original development costs.
- Redevelopment vs. Reengineering (2002 figures)
  - $8-$20 per line of code vs. $2-$5
- Implement good maintenance plan
  - Best case save 15% with maintenance plan.
- US Department of Defense has shown a 150% to 200% savings through reengineering.
Maintenance Costs Over Time with Reengineering
Presenter Program Description

- Presenter program is used to present visual stimulus to animals (cats and monkeys).
- Experiments are performed to contribute to the explanation of how the brain processes visual data through the study of the brain’s bioelectric properties.
- Test ideas about what happens to information contained in a precise visual stimulus after an animal has viewed it.
- Experimental designs in the lab are constantly changing, as are the visual stimulus use in the experiments.
  - Presenter program must be constantly updated.
Presenter Program Description

• Presents very precise visual stimulus to behaving and non-behaving animals.
  – Behaving experiments are terminal and have the animal paralyzed.
  – Non-behaving experiments are non-terminal and involve a surgically implanted magnetic coil around the animal’s eye.

• New stimuli are added for each set of experiments, while old ones are often reused as controls for the experiments.

• Users interact with a graphical user interface to enter the necessary parameters for a particular experiment.
Presenter Architecture

- Video Splitter
- CA-1000 Interface Box
- Microphone
- CONTROL
- PRESENTER
- Primary User Display
- Secondary User Display
- Matrox G400
- Primary Video Signal
- CTR-05 Counter/Timer
- PC-DIO-24 or PCI-6503 Digital I/O Card
- PCI-1200 Data Acquisition Card
- PCI-1200 Data Acquisition Card
- PCI-6071E Data Acquisition Card
- Solenoid Valve (reward)
- Vertical Sync
- Ethernet Hub
- One Degree at 3ft
- Reward Button
- Multi-Electrode Signal
- Horizontal Eye Signal
- Vertical Eye Signal
- Animal Display
Legacy Presenter Program

• The Presenter program was developed by neuroscientists – not computer scientists.
  – Originally developed for a specific purpose.
  – Was not expected to become complex.
  – New requirements added as time passed.
• Its development was haphazard.
• It was modified numerous times, by many people, to add functionality in order to conduct appropriate experiments.
• High turnover of staff in the lab.
• High precision measurements are needed, therefore Presenter must be precise and reliable.
• It has several bugs which affect the results.
  – Some papers were published based on the erroneous data collected!
Legacy Presenter Program

• Started without general design scheme
• Many new additions to code.
• Complex and difficult to:
  – Maintain
  – Familiarize new employees.
    • Poor documentation
  – Add new modules
  – Test
  – Track errors in timely manner – overly consuming
• Time to update the system is excessive due to difficulty in understanding the code and its structure.
Legacy Presenter Program

• The Presenter program has been continuously updated and it is in need of modularization.
  – It is legacy code. The maintenance cost is heavily skewed toward adding new functionality.
  – Adding a new stimulus has become a daunting task.
• The aim is to reengineer the Presenter program:
  – Modularize.
    • Allow new functionality to be added easily.
    – Remove duplicate code.
    • No code sharing or reuse in original code.
  – Object-oriented design.
  – Correct erroneous code.
Presenter GUI
Reengineering – the first step

• The steps for reengineering a piece of legacy software are as follows:
  – Module capture.
    • Documenting and understanding the design of the legacy system.
    • Note problems and duplications within the current system.
    • Can use reverse engineering tools to assist.
  – Problem detection.
    • Identify the parts of the current system which lack flexibility and do not adhere to best practice.
Reengineering – the first step

– Problem analysis.
  • Select a design to address the problems.
– Reorganization.
  • Transformation of the legacy system.
  • Whole system may not need to be transformed at once.
– Change propagation.
  • Test
  • Transition from old system to new system.
  • Use of appropriate tools can help minimize reengineering costs.
Reengineering Using the Horseshoe Model

Architecture Transformation

Base Architecture

Desired Architecture

Architecture Representation

Function-level Representation

Source Text Representation

Legacy Source

Code Structure Representation

New System Source

design patterns & styles

program plans

code styles

Architecture Recovery/Conformance

Architecture Based Development
Rational Rose

• Software engineering tool
• Creates architectural views of software which can be filled in and compiled within rational rose.
• Reverse engineering C++ analyzer
• Create UML diagrams
Unified Modeling Language

- Association
- Dependency
- Composition
- Aggregation
- Inheritance
Original Presenter Architecture

-TGuiWindow

-TAsFieldForm - TGaborTargetGUIForm - TTimingGUI - TMapRFGUI - TEyeCalibGUI

-TReverseCorrelationGUI

-PARADIGM

-SEQUENCE_TYPE

-REPEAT_BLOCKS - SSparseNoiseData

-OBJECT_TYPE - RANDOM_ORDER
Problems with Original Design

• Difficulty understanding and maintaining.
  – Parts written in C – not object-oriented. Other parts written in C++. Inconsistencies.
  – No Modularity
    • Long and repetitive functions that could be further broken down into reusable components (i.e., exploit commonality).
    • High cyclomatic complexity.
    • GUI and runtime code
  – Code not reusable
  – Redundant code
  – Lack of Documentation
  – Global Variables
    • Used heavily – hinders readability and makes maintenance difficult.
  – Poor design
    • “Designed” by non-Computer Scientists.
Problems with Original Design

• These problems kept many necessary items from being added to presenter.
  – i.e., not all the desired functionality was being added. Strong clue that it was time to reengineer the code.
• Much of the redundancy can be removed with a better design.
• A new design also allows common elements to be reused as necessary.
Desired Additions To GUIs
Design Patterns

Patterns emerge when designing software.

- Common problems have been addressed in the past for other problems.
- General design solutions may be addressed to many specific design problems.

• Familiarity with design problems.
  - Difficult to fit software into design problem category.
  - Need general knowledge of current design patterns being used.

• Student reengineering the code was not familiar with design patterns. Success demonstrated that reengineering can be done by any professional software engineer or computer scientist.
Model-View-Controller Paradigm

• The student’s knowledge of design patterns was limited initially in the redesign of the presenter program, however when researching solutions into to the lack of modularity in the GUI, the Model-View-Controller pattern was identified as useful.
• Makes object oriented code with user interfaces quicker to develop and more reusable.
• Separate user interface from functionality used to run the program.
  – View is GUI classes.
  – Model is BasePresenterModule and below.
  – Controller is BaseScreen.
Overall Design and Use of Multiple Inheritance

• All modules starting with “T” means the class name interacts with the user.
  – These are contained within an inheritance hierarchy.
• The modules under BasePresenterModule all contain run time code. GUI presentation and actions not separated.
  – Common elements in the GUI code not identified.
  – The implementation occurs as either an object
    • Bar, spot, square, Gabor, Grating, M-sequence, or Movie
  – or a behaving object which inherits from both the object above it and from Trial.
New Architecture (in full)
New architecture

- The new architecture appears to be as complex (or more) than the original design!
- This is deceptive since the original design contained large subsystems which were related in complex ways.
- The new architecture broke many of these complex classes into smaller constituent classes, and employed inheritance hierarchies to exploit commonality.
Software Metrics

• Function Points,
  – Uses 14 factors based on functionality
  – Here comparing two versions already written for the same functionality so not relevant.
• Halstead’s Complexity Measure,
  – Count operators and operands
  – Lexical and textual complexity
    • Vocabulary, program length, volume, programming effort.
• Cyclomatic Complexity and,
  – Measuring decision statements.
• Problem... the reengineered version has large amount of new functionality which could not be included in measurement.
  – Neuroscientists wanted to continue to increase functionality while the reengineering effort took place!
Software Observations

• ~25% of the total time in reengineering process was spent on understanding the entire systems.
  – It was this \textit{low} only because one of the original architects was available for consultation (this individual’s design capabilities was extremely low).

• Once the original design was understood, a new design could be built so that future functionality could be added with minimal disruption.
  – i.e., Design for change.

• Use of C++ - pragmatic choice since the neuroscientists are to maintain the code in the future.
  – Used object-oriented techniques.
  – Used design patterns.
  – Adopted best practice.
COCOMO II

- Model based on the original work of Barry Boehm in 1981.
- Updated to COCOMO II to take into account new software practices.
  - Uses many subjective variables, guidelines of how to measure these are available.
- Used to estimate cost, development, and schedule when planning new software development activity.
  - Can estimate cost of maintenance over next 5 years.
- Use this approach to estimate maintenance cost for new and old versions of Presenter.
- Was it worth the effort?
Cost maintenance estimate for reengineered vs. original software using COCOMO

![Graph showing maintenance costs for original and reengineered software over years, using COCOMO model.](image_url)
Was Reengineering Worth it?

- Cost of Reengineering estimated without significant time spent on additions.
  - 32 man weeks at $1,000 per week = $32,000
- Difference in cost of maintenance in first year alone for new vs. old system is $51,000.
- Total saving over next 5 years is $518,000.
  - Since the research project didn’t have $500,000 to spend on maintenance in the next 5 years, the reengineering process has effectively extended the life of the research project.
Was Reengineering Worth It?

• The module that was reengineered was mostly contained in one large class or approximately 4,500 lines.
  – Much duplicate code was identified and removed.
  – Object-oriented techniques such as aggregation, composition and inheritance adopted.
  – Design patterns employed.
  – Monolithic code broken into 27 smaller, more logical classes.

• Significant functionality added while performing the reengineering.
  – Reengineered code was about the same size as the original – some of this is due to the overhead of using patterns and more classes.
Was Reengineering Worth It?

• The reengineering process took 32 weeks of effort (part-time).
• Comparing the old and the new architectures, new functionality in the new system requires about 63% less code and can be added far more easily.
• Previous experience with earlier developers hired to work on the Presenter told us that the time required to become familiar with the code was about 4 weeks.
  – It is estimated that it only takes 1 week to become familiar with the new architecture.
Conclusions and Future Work

- Programs in high use but with little design planning such as the Presenter are excellent reengineering candidates.
- For Presenter 75% savings in maintenance costs over the next 5 years.
- Reengineering also identified errors or oversights which were repaired in the new version.
- Eventually after system updates and ensuring all modules are fully validated (experimentally) the Presenter program and the system surrounding it could be marketed to other visual neuroscience laboratories.
Questions?
Bozeman, Montana
class BaseRevCcorr : public BaseScreen
class Trial : public virtual BaseRevCcorr
class Movie : public virtual BaseRevCcorr
class MovieBehavior : public Trial, public Movie
Original Architecture (part)