TJHSST Computer Systems Lab Senior Research Project
An Example of Code Refactoring with Legacy Code in a Flight Model Software
2007-2008

Eric Shi
Mentor: Peter Stassen
The MITRE Corporation

October 18, 2007

Abstract

As code in software ages, it becomes increasingly important for it easily to be maintained and understood by programmers who may work with the code. Modern software may utilize legacy code for the purpose of supporting specific features. Therefore, refactoring techniques must be applied to the code to ease the process of implementing future modifications. Refactoring emphasizes improving the design of an already-written piece of code without changing its function. This project seeks to apply refactoring techniques to a 14-year-old component of the flight model software used at the Air Traffic Managment Lab at the MITRE Corporation's Center for Advanced Aviation System Development.

Keywords: design patterns, flight simulation, legacy code, refactoring

1 Purpose and Scope of Project

The purpose of this project is to refactor and rework three related modules (autoflight, mcp, and fltsim) in the "cockpit" component of an existing flight model program. Refactoring is important in the computer programming industry because it ensures maintainability and readability in software code. The software is changed such that its internal structure is improved, but its function remains the same. The C code used in the modules is 14 years old, so it needs to be updated to meet today's programming standards, such as object-oriented design. This project will develop skills in working with a mature software system, implementing design patterns, and learning to refactor software for maintainability. The result of this project is a single module in C++, which utilizes object oriented design that meets today's programming standards and can be easily maintained in the future.

2 Background and Review of Current Literature and Research

Refactoring is a technique used in the software development cycle to ensure code readability by changing the internal structure of a piece of software without changing its function. There are different types of refactorings, and some examples are found in Martin Fowler's **Refactoring: Improving the Design of Existing Code**. Examples of refactorings are: encapsulate field, extract method, parameterize method, pull up constructor body, push down method, and substitute algorithm. In 1992, William Opdyke wrote his Ph.D. thesis titled "Refactoring Object-Oriented Framework" the first thesis on refactoring. Design patterns, repeatable solutions to common computer programming problems, is another relevant topic in this project. Another goal of this project is to implement some patterns found in **Design Patterns: Elements of Reusable Object-Oriented Software** by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides.

3 Procedure and Methodology

The code used in this project will be C++ written in a Linux environment. Some of the code in the modules was written in C, so the code will be

rewritten to utilize features of C++, namely object-oriented design. This project will attempt to unite the mcp, fltsim, and autoflight modules in the cockpit component of the flight model software to simplify the exchange of information between the modules. This project can be segmented into sections that can be worked on in intervals:

1. Literature review
2. Cataloging of relevant variables
3. Merging into a single modules
4. Transition to an object-orientated designs
5. Testing by subject-matter experts

4 Testing and Analysis

The functionality of the product will be tested in an existing simulation model by subject experts.

5 Expected Results and Benefit to Others

The product is expected to be significantly easier to maintain, updating 14-year old C code with C++ and utilizing common design patterns in software engineering. The performance of the module is expected to be about the same as the original, however. The primary benefit will be to future programmers who intend to implement addition functions into the module.