

# Comprehension and Evolution of Legacy Software

Vaclav Rajlich  
Department of Computer Science  
Wayne State University  
Detroit, MI 48202, USA  
vtr@cs.wayne.edu  
tel. 313-577-5423

## ABSTRACT

This tutorial presents an overview of legacy systems comprehension and evolution. Software comprehension typically consumes more than one-half of the programming effort with the legacy systems. Software evolution is a change in requirements and it is the most common type of change in legacy systems.

## Keywords

Understanding, change propagation, ripple effect, incremental reengineering, redocumentation

## INTRODUCTION

Past emphasis of software engineering has been on development of new software. The reasons are historical. Software is still a relatively new technology, and the current pool of legacy software had to be developed from scratch. However, with the progress of the time, the emphasis is shifting towards the processes dealing with the legacy software.

Legacy systems are often characterized by one or more of the following properties: they were implemented many years ago, their technology became obsolete, their structure deteriorated, they represent a large investment, they contain business rules not available elsewhere, they cannot be easily replaced, and original authors are not available. These properties are the reason why dealing with the legacy systems is difficult.

The most common processes of legacy software are maintenance, evolution, reuse, and reengineering. The purpose of this tutorial is to present an overview of the field of legacy systems comprehension and evolution.

Permission to make digital/hard copies of all or part of this material for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copyright is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires specific permission and/or fee

ICSE 97 Boston MA USA

Copyright 1997 ACM 0-89791-914-9/97/05 ..\$3.50

In order to introduce them, consider the miniprocess of software change, that is the basic building block of all legacy systems processes. It consists of the following phases:

- Request for change
- Understanding the current system
- Implementation of the change
- Propagation of change
- Verification
- Documentation of the change

Software understanding (comprehension) typically consumes more than one-half of the effort. Software evolution is a change in requirements and it is the most common type of change. The tutorial gives an overview of the available techniques and tools for both comprehension and evolution in legacy software.

## COMPREHENSION

Comprehension (or software understanding) is a prerequisite of all legacy systems processes. Only systems well understood can be maintained, evolved, reused, etc.

There are several techniques of program comprehension, of which two are the most common: the top-down technique and the bottom-up technique. The top-down technique is characterized by formulation and verification of hypotheses about the code. Hypothesis is a guess what the design decision was. The basic hypothesis is formed first, and then a chain of subsidiary hypotheses, which depend on it. The chain of hypotheses is verified by inspection of the code. The inspection either confirms the hypotheses, and then they become part of programmer's understanding of the code, or it refutes them and then they are discarded and new ones must be created.

Bottom-up technique for program comprehension is based on the so-called chunking. Chunks are pieces of code with identifiable meaning: algorithms, functions, data structures, representations of domain objects, etc. Chunking consists of aggregation and abstraction. Aggregation is the activity where several components of the code are grouped together. Abstraction is an activity in

which an abstract concept is adopted, which describes the group and gives it a meaning. Both the top-down and bottom-up techniques of comprehension are used by the programmers, and are important for development of comprehension tools and for documentation of legacy systems.

### **Comprehension Tools.**

In order to facilitate the process of comprehension, many software tools have been developed. These tools analyze the code and extract the specific information that the programmer needs. The tools belong to several groups, among them:

*Browsers* help the programmer to navigate through the software, following the dependencies among the software entities. Most often, these are the relationships between the definitions of variables, procedures, types, etc., and their use. The definitions and use may be distributed over distant parts of the software and different files, and the browsers provide quick navigation. This substantially accelerates "flipping" through the code, and lessens the amount of effort necessary for software comprehension. The browsers often display the information graphically, further improving comprehensibility.

*Pattern matching tools* help the programmer with the task of chunking. They analyze the text of the code for patterns which the programmer wants to find. They differ from each other in program representation, and they use different algorithms of pattern matching.

*Slicing tools* reduce the size of the code to be inspected by extracting those parts of program which are related to a specific computation. The program slice is smaller than the whole program, and therefore easier to read and understand.

*Documentation tools* support creation and navigation through the documentation. A well-structured documentation supports the software comprehension techniques.

### **EVOLUTION**

As stated earlier, software evolution is a change in software requirements. It is the most common process for legacy software. Software comprehension is the prerequisite to successful evolution. The other important technique of evolution is the propagation of change.

#### **Propagation of change.**

When a specific part of the code is changed, other parts of the program may be affected and must also change. This is called the change propagation or ripple effect. Change propagates through program dependencies.

The process most commonly used ("change-and-fix process") follows the following scenario: Locate the functionality in the code which is to be changed. Change that functionality, and propagate the change to other constructs through the dependency relations.

For evolution, a top-down process of change propagation is available. It follows the following scenario: The process starts with the top objects. This follows from the fact that the evolution is a change in specifications, and all specifications are tied together in top object, although parts may be delegated to supporting objects. The change propagates top-down to supporting objects.

### **Evolvable Architectures.**

Future evolution can be made easier during software development. The development techniques which make software more evolvable concentrate on either making software more comprehensible, or on shortening the ripple effects.

More comprehensible software is accomplished through the use of coding and design conventions, through the use of constructs with intuitive meaning, through the comprehensible architectures, through documentation (annotations) which respects the comprehension processes, etc.

Shortening of ripple effects is accomplished through parametrization of constructs, designing for ease of extensions and contractions, and design with independent subsystems (higher level structuring), etc.

### **REENGINEERING**

Software reengineering is any activity that improves the quality of legacy software. We are going to limit our exposition to reengineering for comprehensibility and evolvability. The reengineering methods involve restructuring and redocumentation. Incremental reengineering is a process where reengineering is done step-by-step as a part of maintenance miniprocess. It may involve incremental restructuring, which is related to the specific change, and incremental redocumentation, which captures the comprehension gained during the change.

### **EMPIRICAL STUDIES**

The development of comprehension and evolution models, tools, and techniques raises the question of the validity of these techniques. The validation of these techniques is done by empirical studies, and have a form of controlled experiments, case studies, protocol analysis, and other similar methods. The growing body of the work in this area will be reviewed.