Program Comprehension through Data Mining

Ioannis N. Kouris
University of Patras, Dept. of Computer Engineering and Informatics Greece

INTRODUCTION

Software development has various stages, that can be conceptually grouped into two phases namely development and production (Figure 1). The development phase includes requirements engineering, architecting, design, implementation and testing. The production phase on the other hand includes the actual deployment of the end product and its maintenance. Software maintenance is the last and most difficult stage in the software lifecycle (Sommerville, 2001), as well as the most costly one. According to Zelkowitz, Shaw and Gannon (1979) the production phase accounts for 67% of the costs of the whole process, whereas according to Van Vliet (2000) the actual cost of software maintenance has been estimated at more than half of the total software development cost.

The development phase is critical in order to facilitate efficient and simple software maintenance. The earlier stages should be done by taking into consideration apart from any functional requirements also the later maintenance task. For example the design stage should plan the structure in a way that can be easily altered. Similarly, the implementation stage should create code that can be easily read, understood, and changed, and should also keep the code length to a minimum. According to Van Vliet (2000) the final source code length generated is the determinant factor for the total cost during maintenance, since obviously the less code is written the easier the maintenance becomes.

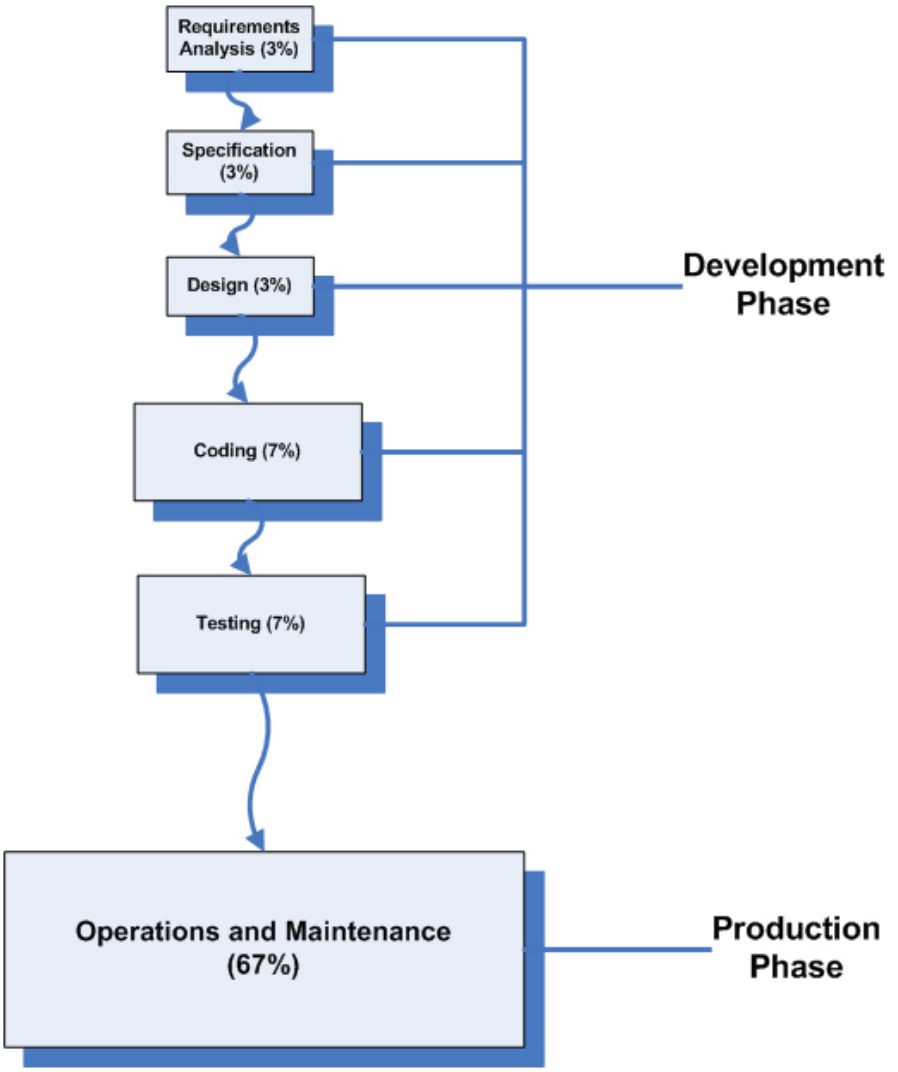
According to Erdil et al. (2003) there are four major problems that can slow down the whole maintenance process: unstructured code, maintenance programmers having insufficient knowledge of the system, documentation being absent, out of date, or at best insufficient, and software maintenance having a bad image. Thus the success of the maintenance phase relies on these problems being fixed earlier in the life cycle. In real life however when programmers decide to perform some maintenance task on a program such as to fix bugs, to make modifications, to create software updates etc. these are usually done in a state of time and commercial pressures and with the logic of cost reduction, thus finally resulting in a problematic system with ever increased complexity. As a consequence the maintainers spend from 50% up to almost 90% of their time trying to comprehend the program (Erdös and Sneed; 1998, Von Mayrhauser and Vans; 1994, Pigoski, 1996). Providing maintainers with tools and techniques to comprehend the programs has become and is receiving a lot of financial and research interest given the widespread of computers and software in all aspects of life. In this work we briefly present some of the most important techniques proposed in the field thus far and focus primarily on the use of data mining techniques in general and especially on association rules. Accordingly we give some possible solutions to problems faced by these methods.

BACKGROUND

Data mining can be defined as the process concerned with applying computational techniques (i.e. algorithms implemented as computer programs) to find patterns in the data. Among others, data mining technologies include association rule discovery, classification, clustering, summarization, regression and sequential pattern discovery (Chen, Han & Yu, 1996).

The use of data mining techniques in program comprehension has been very wide, with clustering being the most popular method. Tjortjis and Layzel (2001) have proposed a tool called DMCC (Data Mining Code Clustering) in order to help maintainers who are not familiar with some software to get a quick overview and speed up the process of getting familiar with it. The specific tool used clustering techniques for grouping together entities that share common characteristics. Mancoridis et al. (1998) have proposed a tool called Bunch that creates a high level decomposition of the structure of a system into meaningful subsystems by using clustering techniques over a Module Dependency Graph. Subsystems provide developers with high-level structural information that helps them navigate through

Figure 1. Software development stages and their relative costs (Zelkowitz, Shaw and Gannon, 1979)



the numerous software components, their interfaces, and their interconnections. Kanellopoulos and Tjortjis (2004) have also used clustering techniques on source code in order to facilitate program comprehension. Their work extracted data from C++ source code, and tried to cluster it in blocks in order to identify logical, behavioral and structural correlations amongst program components.

Other similar works can be found at Anquetil and Lethbridge (1999), Lakhotia (1997) and Tzerpos and Holt (1998). Also an early but thorough overview on software clustering techniques can be found by Wiggerts (1997).

In a significantly smaller degree there has been used also association rules on program comprehension. Tjortjis, Sinos and Layzell (2003) have proposed a technique that inputs data extracted from source code and derives association rules. These rules help the formation of groups of source code containing interrelated entities. Similar work to Mancoridis et al. (1998) but with the use of association rules instead of clustering techniques has been made by De Oca and Carver (1998). In this work by using the ISA (Identification of Subsystems based on Associations) methodology a software system was decomposed into data cohesive subsystems by mining association rules. Use of association rules into program comprehension has been made also by Sartipi, Kontogiannis and Mavaddat (2000) for architectural design recovery.