Using Message Sequence Charts to Accelerate Maintenance of Existing Systems

Nikolai Mansurov[1,2] and Djenana Campara[1]

[1] KLOCwork Solutions
1 Antares Drive, Nepean, K2E 8C4, Ontario, Canada
[2] Institute for System Programming
25 B. Kommunisticheskaya, Moscow 109004, Russia
nick@ispras.ru, djenana@nortelnetworks.com

Abstract. In this paper we describe our experiences in building tools for accelerating maintenance of existing large telecommunications software. We discuss how various maintenance activities can be accelerated by providing developers with the knowledge of the core scenarios of the system, which approximate the intended use cases. We present a static approach to extracting scenarios as source trajectories, by navigating through the source code and capturing the source statements as events. We describe our **PathFinder** tool for static capturing of scenarios. The possibility of static capturing of the core scenarios and their representation as MSCs have benefits in retaining expertise about existing software, in training new personnel, in focusing understanding of legacy software, performing architecture reviews, and in architecture analysis of existing systems. We believe that this approach can contribute to closing the gap between tool support for forward engineering in the so-called "green-field" projects, and maintenance of existing software.

1 Introduction and Background

Recently, time-to-market has become the dominating factor for industrial success, placing enormous pressure on manufacturers and developers to spend less effort on quality and performance, or to find much more efficient means of producing high-quality software for their products and services. Rapidly changing economic conditions, new business paradigms and new technologies have produced a dynamic, global environment and a unique, unprecedented application development challenge. The use of the Internet by both industries and consumers is another dramatic force shaping the future of the global economy.

Traditional development methodologies and tools do not produce applications fast enough to keep up with customer demands and the competitive pressures from more agile companies [9]. Therefore, there is an increasing demand for new methodologies and tools that can speed up the time it takes to design, construct, deploy and maintain applications. New methodologies should support accurate, rapid development and increased developer productivity, enable easy modification and customization of code, content, applications, and business rules [9].

R. Reed and J. Reed (Eds.): SDL 2001, LNCS 2078, pp. 19–37, 2001.

Significant research and development efforts are being invested into acceleration of development and deployment of new systems. Companies like Rational and Telelogic demonstrate, that an up-front model can reap tremendous potential benefits in quality, productivity, reuse and maintenance. Object-oriented analysis, modeling, design and construction tools continue to grow more popular among development organizations [9].

Software components and the component-based development (CBD) paradigm are rapidly reaching a critical mass in terms of adoption by industry. Unlike the modeling-based paradigms, CBD approach relies on the ability to use leverage from a visual paradigm and automation to build and assemble components. Modeling tools can complement CBD tools by providing the means of understanding component internals and making it easier to change their functionality [9].

However, the issue of *accelerating the evolution of existing software* receives considerably less attention. To date, there exists no accepted maintenance methodology, although several such methodologies for development of new systems have already appeared. From the methodology perspective, maintenance is a very complex activity, because it involves a heterogeneous set of short-term processes, potentially with high concurrency. Accelerated maintenance requires a lot more "orchestration" than development of new systems.

Maintenance of existing software includes several activities:

1. corrective maintenance (or defect repair);
2. adaptation maintenance (or modifying the system to accommodate changes in the execution platform;
3. preventive maintenance (or improving the structure of the system to curb architecture erosion); and
4. evolution (or enhancement, adding new features) - designing a new feature of an existing system can be more difficult than designing an equivalent system from scratch, because all forward engineering decisions are constrained by legacy decisions.

Therefore, there are some problems, specific to maintenance, which are not present during development of new systems:

- high-level design decisions need to be rediscovered;
- higher volume of information that needs to be considered (understanding of legacy decisions in unfamiliar source code);
- "expertise" walks away as key developers change jobs;
- need for training new personnel
- multi-site collaborative environments, as for example maintenance is outsourced;
- change resistance due to architecture erosion (usually it is progressively more difficult to change the system).

Complexity of maintenance activity and the lack of methodologies have created a situation, where there exists a significant gap in tool support between