Available online at www.sciencedirect.com
ScienceDirect
The Journal of Systems and Software 81 (2008) 1738–1753
The Journal of Systems and Software
www.elsevier.com/locate/jss

Early quality monitoring in the development of real-time reactive systems ☆

O. Ormandjieva [a,*], V.S. Alagar [a], M. Zheng [b]

[a] Department of Computer Science and Software Engineering, Concordia University, EV3.165, 1455 de Maisonneuve West, Montreal, Quebec, Canada H3G 1M8
[b] Department of Computer Science, University of Wisconsin-LaCrosse, La Crosse, WI, USA

Received 27 February 2006; received in revised form 20 December 2007; accepted 20 December 2007
Available online 18 January 2008

Abstract

The increasing trend toward complex software systems has highlighted the need to incorporate quality requirements earlier in the development cycle. We propose a new methodology for monitoring quality in the earliest phases of real-time reactive system (RTRS) development. The targeted quality characteristics are functional complexity, performance, reliability, architectural complexity, maintainability, and test coverage. All these characteristics should be continuously monitored throughout the RTRS development cycle, to provide decision support and detect the first signs of low or decreasing quality as the system design evolves. The ultimate goal of this methodology is to assist developers in dealing with complex user requirements and ensure that the formal development process yields a high-quality application. Each aspect of quality monitoring is formalized mathematically and illustrated using a train–gate–controller case study.

Keywords: Real-time reactive systems; Quality measurement model; Early reliability assessment; Markov chains; Information theory; Complexity; Maintainability; Performance

1. Introduction

The number of mission-critical software applications with a high cost of failure has increased exponentially in recent years, and the need for quality control early in the development process is now greater than ever. This is especially true of real-time reactive systems (RTRS), which are inherently complex and continuously interact with their environment (Harel and Pnueli, 1985). (In the context of this paper, the term "system" is restricted to mean a "software system".) RTRS systems are used for patient control, air traffic control, nuclear power plants, "intelligent" highways, and telecommunication. Factors contributing to RTRS complexity include time constraints on stimuli and responses, strict safety requirements, complex sequencing of events, and increasingly uncertain environments.

Research Goal. For the purposes of software quality monitoring, the development and evolution of an RTRS has a number of important requirements (Asling Microsystems, Inc., 2007):

- Responsibility for RTRS software quality is not confined to the final product testing stage. Quality control should be applied continually throughout the system's initial development stages.
- A pre-defined, quantified level of reliability is demanded.
- High-quality testing is also required to guarantee the safety and reliability of the system.

☆ This work is supported by Grant from Natural Sciences and Engineering Research Council, Canada to the first author.
* Corresponding author. Tel.: +1 514 848 2424x7810; fax: +1 514 848 28 30.
E-mail address: ormandj@cse.concordia.ca (O. Ormandjieva).

0164-1212/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2007.12.808

- Any characteristics of the system critical to performance or safety, such as timing constraints, must operate within quantifiable limits.
- RTRS software places additional and exacting demands on the software quality monitoring. The underlying quality measurements must be theoretically valid, accurate, and repeatable.

The purpose of this paper is to describe a formal, robust quality monitoring methodology which guarantees that all these requirements are met.

Approach. This paper shows how to monitor quality in the early phases of development by building measurement mechanisms into the process. The goal is to provide *continuous* feedback on the effect of development activities on the quality goals listed above, which are defined in terms of user requirements, scenarios, and other artifacts. In this manner quality requirements can be incorporated into the RTRS at the outset, rather than by adjusting the code once the system is operational and testing begins. Needless to say, the latter solution is far from cost-effective.

We have derived a hierarchical quality measurement model (IEEE Std 1061, 1998; Fenton and Pleefeger, 1998) to support the early phases of RTRS development, along with a monitoring mechanism to collect theoretically valid measurements, analyze them, and provide timely feedback on the phase in which the data originated. Our quality model incorporates functional complexity, performance, reliability, architectural complexity, maintainability, and test coverage (see Fig. 1). We also include an algorithm for optimizing a budget-limited selection of test cases. This choice of quality characteristics is inspired by the ISO/IEC 9126-1 international standard (International Standard ISO/IEC, 2007).

The measurement methods described in this paper is independent of the implementation and allows for quality monitoring during the specification, design, and scenario-based testing processes. Furthermore, this independence allows the impact of a change to the evolving RTRS to be assessed before the change is authorized.

Contributions. The proposed RTRS quality measurement model includes several innovations: (i) integration with a formal development framework; (ii) a mathematical basis for the quality measurements; and (iii) its applicability to early phases of the development. To the best of our knowledge, the software industry currently lacks a continuous quality monitoring mechanism with these advantages. We believe that the proposed model will also increase the quality of requirements specification, system design, and scenario-based testing, thereby decreasing the probability of development errors and increasing the dependability of the final product.

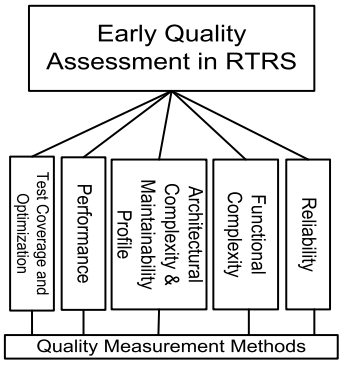

Fig. 1. RTRS quality model.

The monitoring mechanism continuously collects and processes quality measurements, and uses the data to provide timely feedback on whether quality requirements are being met. The *railroad crossing* (Heitmeyer and Mandrioli, 1996) example, a benchmark case in the real-time systems community, is used throughout this paper to illustrate quality measurements and the role of quality monitoring in each phase.

Organization of the paper. The rest of the paper is organized as follows: Section 2 introduces a RTRS process model and development environment. This will serve as a generic setting for the paper, and is used to define the context for the quality monitoring model. Section 3 briefly reviews related work on quality modeling in RTRS, and describes our proposal in this context. Sections 4–6 develop the model in more detail and illustrate quality measurement methods using the case study; related work is also mentioned here where applicable. The paper concludes in Section 7 with a summary of major contributions and suggestions for future research.

2. Development environment

This section introduces the formal model and a RTRS development environment. The quality monitoring process is explained in the context of a research test bed called TROMLAB (Alagar, 2001). This test bed uses a timed, labeled transition system to construct reactive objects. The specifications and design of the RTRS can be composed, edited, and refined using the visual tool Rational Rose, which automatically translates changes to formal notation.

The process requires a formal model of the environment to be integrated with the system's elements. An analysis phase establishes interaction patterns between surrounding objects and the system, then defines their associated processing and timing requirements. Functional and timing requirements for the RTRS itself can then be identified and translated into formal specifications. The specifications are parsed automatically, checked for appropriate syntax, and mapped to an internal representation. This representation is used to simulate the system's behavior during the design and development phases, enabling a systematic validation of the system before it is implemented. Scenario-based test cases are automatically generated from the specifications.