A TAXONOMY OF DYNAMIC SOFTWARE-FAULT MONITORING TOOLS

NELLY M. DELGADO

Computer Science Department

APPROVED:

Ann Q. Gates, Ph.D., Chair

Steven M. Roach, Ph.D.

Scott Starks, Ph.D.

Charles H. Ambler, Ph.D.
Dean of the Graduate School

To Dr. Gates

A TAXONOMY OF DYNAMIC SOFTWARE-FAULT MONITORING TOOLS

by

NELLY M. DELGADO, B.S.

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Computer Science Department

THE UNIVERSITY OF TEXAS AT EL PASO

December 2001

ACKNOWLEDGEMENTS

I would first like to acknowledge my advisor, Dr. Ann Gates and my committee members, Dr. Steven Roach and Dr. Scott Starks, who graciously gave of their time, efforts, and encouragement. Their support was invaluable.

I gratefully acknowledge my family and friends – your love and laughter sustained me. I especially want to thank the individuals who were once my classmates and teachers and who are now my friends. Without their camaraderie, I would not have done so well, nor enjoyed my studies so much.

As an undergraduate, I received the Elayne and Julian Bernat Presidential Scholarship and was funded as a research assistant under NASA grant number NAG2-1012 and NSF grant number EIA-9522207. As a graduate, I was the recipient of an MS Engineering Fellowship from the National Consortium for Graduate Degrees for Minorities in Engineering and Science and a scholarship from the National Science Foundation. I appreciate the organizations and individuals who made my education and research experience possible.

ABSTRACT

Many approaches and tools have been proposed for run-time monitoring with the purpose of detecting, diagnosing, and recovering from software faults. There is a plethora of information surrounding run-time monitors that check for correct behavior in software systems. In this work, a representative sample of the diverse range of run-time software monitoring research is classified based upon the basic elements that are necessary for developing such a system. The taxonomy categorizes the various run-time monitoring research in a manner that supports identification of (1) classes of assurance properties that can be specified and applied, (2) required infrastructure, (3) support provided to the user, and (4) types of applications that are targeted by a given monitor. The last comprehensive survey of the field of run-time software monitoring was published in 1981 by Plattner and Nievergelt. There have been significant advances in the field since this time and, although other surveys have classified the advances, none have been comprehensive.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Many approaches and tools have been proposed for run-time monitoring with the purpose of detecting, diagnosing, and recovering from software faults[1]. There is a plethora of information surrounding run-time monitors that check for correct behavior in software systems. The last comprehensive survey of the field of software-fault monitoring [44] is twenty-years old, and there have been significant advances in the field since then. Unlike the previous work that examined concepts, goals, and limitations of monitors, and other work [50] that considers application-specific monitors, the taxonomy presented in this thesis provides a classification based on the application and implementation of monitors that are used for software-fault detection, diagnosis, and recovery. Specifically, the thesis does not include run-time monitoring used for purposes such as program debugging, detection of hardware failures, measuring of system performance, or program optimization.

1.1 Significance of Work

Taxonomies structure the body of knowledge in a specific area and serve as a resource for decision-making. This work presents a taxonomy for run-time monitoring approaches and tools. The taxonomy categorizes the various run-time monitoring research in a manner that supports identification of (1) classes of assurance properties

---

[1] *Software-fault* refers to a system state that may lead to a failure if not corrected. Other definitions refer to this as an error.

that can be specified and applied, (2) required infrastructure, (3) support provided to the user, and (4) types of applications that are targeted by a given monitor. The taxonomy is applied to a wide range of run-time monitoring systems.

A user can employ this taxonomy to:

- understand and classify the state-of-the-art in run-time monitoring approaches and techniques;
- provide a basis for discussion about these approaches and tools;
- identify kinds of assurance properties that can be specified and applied to a given system; and
- identify existing run-time monitoring approaches and tools that can be applied to a system.

1.2 Thesis Organization

The thesis is organized as follows. Chapter 2 gives an overview of fault detection and run-time monitoring and shows how monitoring approaches can be used for fault detection. In addition, it defines the scope for the run-time monitors that are examined in this thesis. Chapter 3 presents the taxonomy used to classify run-time monitoring approaches and tools. Chapter 4 applies the taxonomy to representative approaches and tools and presents a summary of each. Chapter 5 discusses related work that is outside the scope of the examined run-time monitors. The thesis ends with a summary in Chapter 6.

Chapter 2
FOUNDATION

This chapter describes software verification approaches, introduces approaches and tools used for run-time monitoring, and defines the elements for run-time monitoring tools considered in this paper. This chapter provides the foundation for understanding the placement of approaches and tools in the taxonomy presented in Chapter 3.

2.1 Software Verification

The purpose of software verification is to provide assurance of correct program operation. Figure 1 gives an overview of various methods for providing evidence of correct program execution during a software system's lifecycle. In is important to note that software verification approaches and tools have different strengths and weaknesses, and several of these approaches and tools should be utilized in order to provide a high degree of assurance of correct program operation. This section places run-time monitoring approaches in context with other verification approaches.

Walkthroughs and *inspections* are two types of review activities. The benefits of reviews are that errors can be caught early in the software life-cycle and the earlier that errors are caught, the less expensive they are to fix. The downside is that these are manual, time-intensive activities.

Program synthesis and *correctness proof* approaches address program correctness by constructing provably correct software [7, 15, 37]. Program synthesis approaches include deductive synthesis and transformational synthesis. Although systems constructed

utilizing these approaches produce software of high quality, the disadvantage to these approaches are the difficulty and cost associated with formally specifying knowledge about the problem domain and the program's behavior. Another drawback is the likelihood of introducing errors in the specifications. Finally, like model checkers, tools that are used in program synthesis can also suffer from state explosion problems.

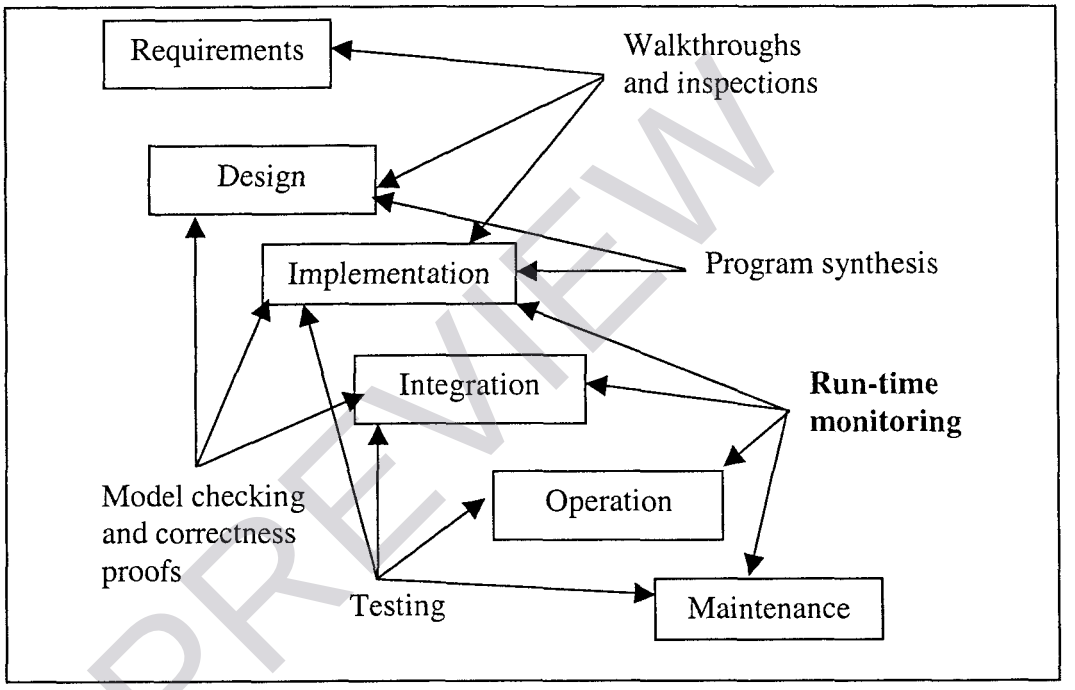

Figure 1. Verification techniques across a system's life-cycle.

Model checking [7, 26, 48] is a technique used to search for faults in a program by testing a desired property in a model of the program. If faults exist in the model, model checking can be used to detect them by testing properties in each state of the complete state-space expansion of the model. Two of the difficulties associated with model checkers are the creation of the model from a program and the infeasibility of searching a large number of states.