

A Hierarchical Classification for Software Health Indicators

Alexander Lau, Barry Pekilis, Naghmeh Ghafari, Rudolph Seviora
Bell Canada Software Reliability Laboratory, University of Waterloo
Waterloo, Ontario, Canada, November 2002
{alexlau, bpekilis, nghafari, seviora}@swen.uwaterloo.ca

I. INTRODUCTION

Experience shows that external failures of software systems are often preceded by deterioration in their internal state (i.e. an error). An error is defined as the difference between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition [1]. For software systems that are designed to degrade gracefully, the capability to provide a statistical indication of their internal well-being or *health* would be very valuable. For example, providing operators with advance warning allows them to take pre-emptive action before a major operational disruption occurs. *Software health monitoring* is an approach that strives for the early detection of internal errors in operational software systems [2]. This approach employs *software health indicators* to monitor particular facets of a target program's execution. Each indicator derives its base monitoring information from one or more software sensors that collect data from specific parts of the internal program state. This paper proposes a hierarchical classification for software health indicators. The classification organises indicators into categories based on the number of information sources examined, as well as on how those sources relate to each other. The classification can be used to identify appropriate indicator classes for detecting errors and can serve as a guide for retrofitting health indicators for monitoring existing software.

During the course of this research, the authors performed an extensive literature review of software monitoring systems and error detection techniques such as the use of assertions. The authors also gained hands-on experience deploying health indicators into several different target systems. This work resulted in the evolution of patterns for health indicators with common functionality and produced this classification. To date, more than 30 indicator classes have been identified under five main categories. The classification has provided a better understanding of the structure and diversity of indicators for health monitoring and helped the authors identify deficiencies in monitoring coverage. The classification can accommodate new indicator classes by extending existing categories or adding new ones.

II. HIERARCHICAL CLASSIFICATION

In the context of health monitoring, the entire state of a software system, at any given moment in time, may be represented by all its data in conjunction with the execution positions of all its threads. While it might be desirable to monitor the entire program state, the computational cost makes this approach impractical. Instead, the entire program state

is decomposed into *state elements*, where each state element represents a unique part of the program state at a given point in time.

The basis of the hierarchical classification is the number of state elements collected by a health indicator as well as how these elements are examined. The top level, as shown in Fig. 1, is divided according to the number of state elements being examined, namely health indicator classes that examine a single state element, and those that examine multiple state elements. The second level is divided according to how those state elements, for a given indicator, are examined.

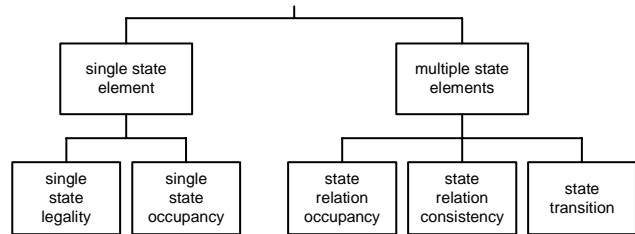


Figure 1. Hierarchy of Software Health Indicators

A. Single State Element

The simplest class contains indicators that examine a single state element.

Single State Legality – This category contains indicators that examine the content of one state element for errors, inconsistencies, or specific conditions. Fig. 2a shows a state element at a given point in time. For example, assertions made on one state element are classified here, such as checking if the content of a numeric age field is positive.

Single State Occupancy – This category contains indicators that measure the amount of time the content of one state element remains unchanged. Fig. 2b shows a state element with unchanged content that is held for a period of time. An example in a traffic light controller is checking if a light remains amber for the required amount of time.

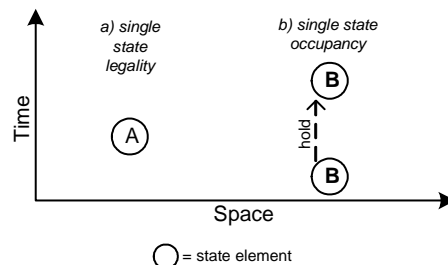


Figure 2. Health Indicators Involving One State Element

B. Multiple State Elements

A more complex class contains health indicators that examine the *relation* between two or more state elements. Often relations involving multiple state elements can be decomposed into a number of equivalent pair-wise relations. For presentation simplicity, we consider relations between two state elements.

State Relation Consistency – This category contains indicators that examine the consistency relation between two state elements. A *consistency relation* is a logical association between two or more specific state elements. They exist as a result of information redundancy within the program state and can often be identified using the design specification. Fig. 3a shows such a relation between two state elements at a given point in time. For example, when transferring monies between accounts, the amount debited from one account must equal the amount credited to the other account. Consistency checking only provides assurance that the relation is legal and not whether the contents of the state elements are correct.

State Relation Occupancy – This category contains indicators that measure the amount of time a relation between two state elements remains unchanged, as shown in Fig. 3b. For example, in a computer aided dispatch system, the amount of time a mobile unit is assigned to a given incident must be within a specified limit.

State Transition – This category contains indicators that examine the change between two state elements. As a software system executes, it moves from one state to another due to external stimulus or through the execution of its program instructions. When the content of a state element changes, a transition is said to have occurred from one state element to another, as shown in Fig. 3c. For example, in a telephone exchange, a transition may occur from the ‘waiting for first digit’ state to the ‘waiting for second digit’ state. Transition checking only provides assurance that the transition is legal and not whether the contents of the state elements are correct.

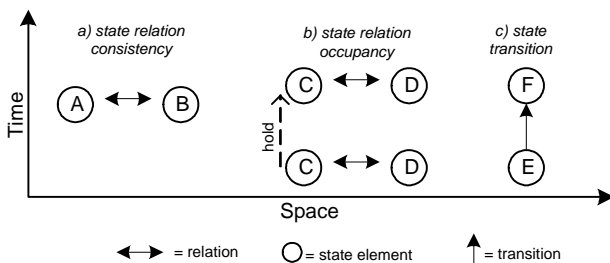


Figure 3. Health Indicators Involving Two State Elements

III. APPLICATION

The hierarchical classification organises software health indicators into classes and provides a reference for identifying appropriate indicators for detecting errors, inconsistencies, or specific conditions. In the case of new software systems, design specifications may be used to identify important state elements and help developers select suitable indicators for software health monitoring. For systems in their mainte-

nance phase, the classification can be used for selecting indicators to capture specific information and errors. For example, the hierarchical classification may be used in conjunction with the *classification problem solving* paradigm [3]. Suppose during maintenance, we need to address a recurring error manifesting as a failure due to *lack of resources*. Using the paradigm, the concrete problem, as illustrated in Fig. 4, is abstracted to the general problem class of *resource leaks*. There is a *heuristic relation* between the problem and solution classes typically based on either an implicit model or empirical experience. In this case, the solution class is a specific indicator category from the hierarchical classification. Resource leaks may be captured as corruption between the resource provider and one or more clients. Therefore the state relation consistency category is selected as an appropriate solution class. Finally, refinement leads to a concrete solution using a specific indicator to monitor the resource manager and the individual clients in a pair-wise manner, such as a resource utilization counter.

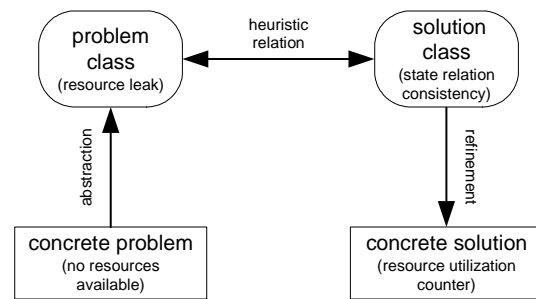


Figure 4. Classification Problem Solving

IV. CONCLUSION

This paper presented a hierarchical classification for health indicators used in software health monitoring. The classification was organised into five categories based on the number of information sources examined, as well as on how those sources relate to each other. The classification helped the authors identify deficiencies in monitoring coverage by providing a better understanding of the structure and diversity of indicators. The classification can be used as a reference for identifying appropriate indicators for detecting specific errors or conditions. An example using the classification in conjunction with the classification problem solving paradigm to select an appropriate indicator was presented.

ACKNOWLEDGEMENT

This work was supported by the Ontario Ministry of Energy, Science, and Technology, as well as the Natural Sciences and Engineering Research Council of Canada.

REFERENCES

- [1] IEEE 610.12-1990: Standard glossary of software engineering terminology. IEEE, 1990.
- [2] J. Thai, B. Pekilis, A. Lau, R. Seviara. Aspect-Oriented Implementation of Software Health Indicators. Asia Pacific Software Engineering Conference. IEEE, 2001. 96~104.
- [3] W. J. Clancey. Classification Problem Solving. National Conference on Artificial Intelligence. AAAI, 1984. 49~55.