In the 1990s, there is an emphasis on finding ways to lower software cost and improve quality. Thus, it is very important to quantify and measure factors, such as software complexity, which have been shown to affect cost and quality. Researchers have defined many software complexity measures, and have exploited them to identify fault-prone program modules, to predict the number of faults that testing and operation will reveal, or to assess maintainability. Information theory based software measures are attractive because they quantify, with a standard unit, the amount of information in an abstraction of a program. The unit of measure is a bit. The purpose of this paper is to survey the state of the art of applications of information theory to software measurement, beginning in 1972. Information theory based measures have been applied to most phases of the development lifecycle. However, there have been relatively few papers. Most measures have not been empirically validated. One can conclude that the field is in its infancy. Relevant concepts of information theory are briefly described, and tables summarize the references from various perspectives. Since the amount of research, thus far, has been very limited, researchers will find numerous opportunities to validate, refine, and improve the measures presented here. Such research should give future practitioners useful software measures for each phase of the lifecycle.

Keywords: software metrics, software measurement, information theory, entropy, software complexity, software quality modelling

1. Introduction

Curtis (1979) describes complexity as 'an attribute of the interaction between two systems that describes the resources one system will expend in interacting with the other system'. In this sense, software complexity is an attribute of the interaction between a development organization and a software product, describing the resources that the organization will expend in developing and maintaining the software product. In the 1990s, there is an emphasis on finding ways to lower software cost and improve quality. Thus, it is very important to quantify and measure factors, such as software complexity, which have been shown to affect cost and quality (Porter and Selby, 1990; Troy and Zweben, 1981).

Measurement is fundamental to the engineering of high quality software (Basili et al., 1986). Static software complexity measures quantify attributes of software products that are related to difficulty in human understanding. As the static complexity of a software product increases, so
does the probability of human error in understanding the product. Researchers have defined many software complexity measures (Zuse, 1991), and have exploited the relationship between these measures and software quality measures by employing software quality models to identify fault-prone program modules (Porter and Selby, 1990; Munson and Khoshgoftaar, 1992a; Khoshgoftaar et al., 1994), to predict the number of faults that testing and operation will reveal (Troy and Zweben, 1981; Khoshgoftaar and Munson, 1990), or to assess maintainability (Henry and Wake, 1991; Khoshgoftaar and Munson, 1992a; Zhuo et al., 1993).

Any single primitive software measure that measures a software development product is too narrow in scope to capture fully the intuitive concept of complexity (Fenton, 1992). Research has found software complexity to be a multidimensional concept (Munson and Khoshgoftaar, 1989). This multidimensionality has resulted in a plethora of primitive measures, each addressing an aspect of complexity along one dimension. Many of these measure very similar concepts, and thus, are highly correlated. High correlation, in turn, often makes them difficult to use in software quality models. Thus, one must carefully select primitive measures that can be shown to contribute significantly in one of the various dimensions of complexity (Munson and Khoshgoftaar, 1991; 1993).

Controlling cost and quality using a multitude of primitive complexity measures has proven difficult. Attempting to address this, many synthetic complexity measures have been proposed that combine primitive measures, often with incomparable units, seeking to reduce the multidimensionality to a single figure of merit. Some are theoretically based (Halstead, 1977), and some are empirically based (Munson and Khoshgoftaar, 1990; 1992b; Khoshgoftaar et al., 1993).

Information theory based software measures are attractive because they quantify, with a standard unit, the amount of information in a program along a dimension. Consider a program to be a message received by a programmer. Information is informally defined as the degree of surprise when a message is received. Entropy is a measure of the average information per message unit (Shannon and Weaver, 1949). The unit of information is a bit.

Most information theory based software measures are motivated by the intuitive notion that the more information a software product contains, the more complex it appears to a software engineer, and hence the more difficult to understand. If software is more difficult to understand, then there is increased risk of higher cost and lower quality. If information content is synonymous with software complexity, and if a goal of software engineering is to minimize complexity, then it follows that measuring the amount of information is prerequisite to cost-effective engineering of quality software.

A number of applications of information theory to software measurement have appeared in the literature over the years, addressing various phases of the development lifecycle. However, only a few papers have been published for each phase. Except for an early review by Mohanty (Mohanty, 1979), previous surveys did not give attention to this class of measures (Cook, 1982; Harrison, 1984; Waguespack and Badlani, 1987; Côté et al., 1988).

The purpose of this paper is to survey the state of the art for applications of information theory to software measurement, in terms of extent, depth, maturity, and opportunities, beginning in 1972. Relevant concepts of information theory are briefly described below, and tables summarize the references from various perspectives. Table 1 lists the measures covered by this survey. Discussion of the measures and the tables are organized by phases of the lifecycle.