Building Empirical-Based Knowledge for Design Recovery

Hee Beng Kuan Tan
Nanyang Technological University, Singapore

Yuan Zhao
Nanyang Technological University, Singapore

INTRODUCTION

Although the use of statistically probable properties is very common in the area of medicine, it is not so in software engineering. The use of such properties may open a new avenue for the automated recovery of designs from source codes. In fact, the recovery of designs can also be called program mining, which in turn can be viewed as an extension of data mining to the mining in program source codes.

BACKGROUND

Today, most of the tasks in software verification, testing, and re-engineering remain manually intensive (Beizer, 1990), time-consuming, and error prone. As many of these tasks require the recognition of designs from program source codes, automation of the recognition is an important means to improve these tasks. However, many designs are difficult (if not impossible) to recognize automatically from program source codes through theoretical knowledge alone (Biggerstaff, Mitbander, & Webster, 1994; Kozaczynski, Ning, & Engberts, 1992). Most of the approaches proposed for the recognition of designs from program source codes are based on plausible inference (Biggerstaff et al., 1994; Ferrante, Ottenstein, & Warren, 1987; Kozaczynski et al., 1992). That is, they are actually based on empirical-based knowledge (Kitchenham, Pfleeger, Pickard, Jones, Hoaglin, Emam, & Rosenberg, 2002). However, to the best of our knowledge, the building of empirical-based knowledge to supplement theoretical knowledge for the recognition of designs from program source code has not been formally discussed in the literature.

This paper introduces an approach for the building and applying of empirical-based knowledge to supplement theoretical knowledge for the recognition of designs from program source codes. The first section introduces the proposed approach. The second section discusses the application of the proposed approach for the recovery of functional dependencies enforced in database transactions. The final section shows our conclusion.

MAIN THRUST

The Approach

Many types of designs are usually implemented through a few methods. The use of a method has a direct influence on the programs that implement the designs. As a result, these programs may have some certain characteristics. And we may be able to recognize the designs or their properties through recognizing these characteristics, from either a theoretical or empirical basis or the combination of the two.

An overview of the approach for building empirical-based knowledge for design recovery through program analysis is shown in Figure 1. In the figure, arcs show interactions between tasks. In the approach, we first research the designs or their properties, which can be recognized from some characteristics in the programs that implement them through automated program analysis. This task requires domain knowledge or experience. The reason for using design properties is that some designs could be too complex to recognize directly. In the latter case, we first recognize the properties, then use them to infer the designs. We aim for characteristics that are not only sufficient but also necessary for recognizing designs or their properties. If the characteristics are sufficient but not necessary, even if we cannot infer the target designs, they do not imply the nonexistence of the designs. If we cannot find characteristics from which a design can be formally proved, then we will look for characteristics that have significant statistical evidence. These empirical-based characteristics are taken as hypotheses. With the use of hypotheses and theoretical knowledge, a theory is built for the inference of designs.

Secondly, we design experiments to validate the hypotheses. Some software tools should be developed to automate or semiautomate the characterization of the properties stated in the hypotheses. We may merge multiple hypotheses together as a single hypothesis for the convenience of hypothesis testing. An experiment is designed to conduct a binomial test (Gravetter & Wallnau, 2000) for each resulting hypothesis. If altogether we have k hypotheses denoted by $H^1,\ldots, H^k$ and we would like the probability of validity of the proposed design recovery to be more than or equal to q, then we must choose $p_1,\ldots, p_k$ such that $p_1,\ldots, p_k \geq q$. For each hypothesis H^j ($1 \leq j \leq k$), the null and alternate hypothesis of the binomial test states that less than p_j*100% and equal or more than p_j*100%, respectively, of the cases that H^j holds. That is:

H^j_0 (null hypothesis): probability that H^j holds < p

H^j_1 (alternate hypothesis): probability that H^j holds ≥ p

For the use of normal approximation for the binomial test, both np_j and $n(1-p_j)$ must be greater than or equal to 10. As such, the sample size n must be greater than or equal to max ($10/p_j$, $10/(1-p_j)$). The experiment is designed to draw a random sample of size n to test the hypothesis. For each case in the sample, the validity of the hypothesis is examined. The total number of cases, X, that the hypothesis holds is recorded and substituted in the following binomial test statistics:

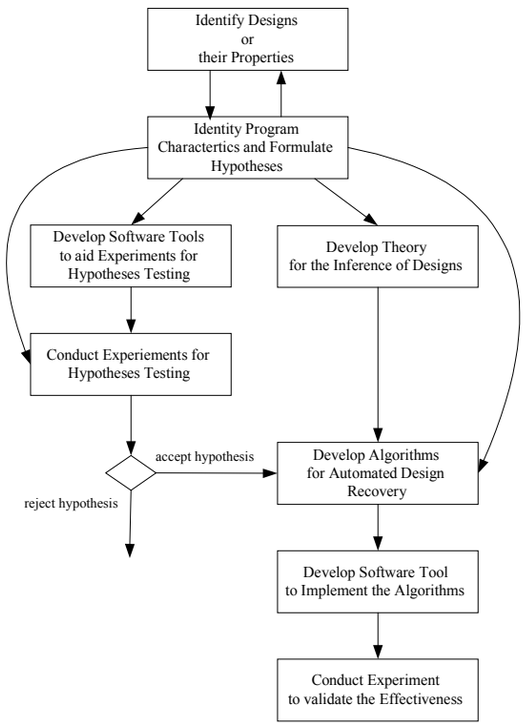
$$z = \frac{X/n - p_j}{\sqrt{(p_j(1-p_j)/n)}}$$

Let α be the Type I error probability. If z is greater than z_α, we reject the null hypothesis; otherwise, we accept the null hypothesis, where the probability of standard normal model for $z \geq z_\alpha$ is α.

Thirdly, we develop the required software tools and conduct the experiments to test each hypothesis according to the design drawn in the previous step.

Fourthly, if all the hypotheses are accepted, algorithms will be developed for the use of the theory to automatically recognize the designs from program source codes. A software tool will also be developed to implement the algorithms. Some experiments should also be conducted to validate the effectiveness of the method.

Figure 1. An overview of the proposed design recovery



Applying the Proposed Approach for the Recovery of Functional Dependencies

Let R be a record type and X be a sequence of attributes of R. For any record r in R, its sequence of values of the attributes in X is referred as the *X-value* of r. Let R be a record type, and X and Y be sequences of attributes of R. We say that the *functional dependency (FD), $X \rightarrow Y$ of R, holds at time t,* if at time t, for any two R records r and s, the X-values of r and s are identical, then the Y-values of r and s are also identical. We say that the *functional dependency holds* in a database if, except in the midst of a transaction execution (which updates some record types involved in the dependency), the dependency always holds (Ullman, 1982).

Many of the world's database applications have been built on old generation DBMSs (database management systems). Due to the nature of system development, many