Summary of Dissertation

Research on Construction and Applications of Formal Specification Component Attributes to Support Specification-Based Software Development

(仕様に基づくソフトウェア開発における形式仕様のコンポーネント属性の構築と応用に関する研究)

Li Jiandong

Most, if not all, of the phases in software development benefit from the use of formal specification. However, there still exist some challenges or problems in formal specification-based software development. The identified challenges are:

(1) Components of a formal specification do have attributes and relationships hided in the specification, but they are not explicitly derived and described for making better sense of specification understanding and supporting subsequent development tasks.

(2) How to prevent software faults during programming remains a challenge. Since software faults are costly to find and remove from programs, effective and proactive fault prevention approaches in coding are in high demand.

(3) Little work on automatically building trace links between formal specifications and code (formal-specification-to-code) is done. Building effective trace links between formal specifications and their implementation is essential for conformance verification and program maintenance.

(4) Formal specification-based code inspection (FSBCI) is a static technique for program verification. However, the program reading techniques used in the existing FSBCI methods, such as checklist-based reading, suffer from limited guidance and support for inspectors on fault detection.

To increase the benefit of formal specification to software development and address these problems, in this research, I first design multidimensional attributes for various formal specification components with respect to their structures and relationships with other components. Then, I put forward an approach to transforming a formal specification into a knowledge graph, which stores the extracted multidimensional attributes of specification components. Using the formal specification component attributes as a firm basis, I also put forward a fault prevention method to enhancing productivity and reducing the fault introduction risk during programming, an automated formal-specification-to-code trace links establishment approach to supporting conformance verification and program maintenance, and a formal specification component attributes-based code inspection (FSCABCI) approach to detecting requirements-related faults.

In detail, my research benefits formal specification-based software development from the following aspects.

1. Knowledge graph construction for SOFL formal specifications.

I propose a top-down approach to constructing knowledge graph from formal specifications. The knowledge graph is essentially a set of Subject-Predicate-Object (SPO)

triples that is compatible with RDF data model and means that subject *S* has property *P* with value *O*. To model, organize and store the components of the formal specification, and represent the multi-dimensional attributes of components and their relationships using a knowledge graph will make the specification easy to use and understand and provide a foundation for formal specification-based tasks. The top-down approach means that the ontology of a knowledge graph is defined first, and then instances are added to the knowledge graph. The construction process includes the following steps:

(1) creating an ontology by using the seven-step method published by Stanford University and the ontology editing tool Protégé,
(2) designing the E-R diagram of the relational database based on the created ontology,
(3) extracting and storing attribute and relationship information about constituent components of a formal specification in the relational database,
(4) mapping ontology to its instances and relational data to resource description framework (RDF) triples,
(5) displaying knowledge graph via Neo4j graph database.

Further, a case study on knowledge graph construction from the SOFL formal specification of an ATM system is conducted to demonstrate the feasibility of our approach in practice. In particular, we construct an ontology in the domain of SOFL formal specifications, which contains 24 classes (concepts), 40 object properties (relationships between concepts), and 47 data properties (properties of classes). The created knowledge graph for the ATM specification encapsulates about 1066 RDF triples totally.

2. Fault prevention in formal specification-based programming.

To mitigate the effect of the faults introduced into the programs and thus reduce the time and cost in fault detection, fault fixing and software retesting, we are stimulated to work out an effective and systematic fault prevention approach for formal specification-based programming. After identifying requirements-related faults during the transformation from formal specifications to programs and analyzing their root causes, I propose a fault prevention approach.

To prevent faults from being generated during coding, my method first automatically identifies the constitutive components in the specification and infers their appropriate implementation order through component dependence analysis. Then, using predefined transformation patterns, the automatic generation of code fragments for the components (except the operations) in the specification is enabled. In particular, my method decomposes the pre- and postconditions of an operation in the specification into its equivalent static, single-assignment form, which is a set of instructions in the form:

$$result = operator(operand1, \ldots, operandN)$$

To enable the automatic transformation of these instructions into code, I also build a database that maps derived instruction to its corresponding code fragment. Finally, the programmer must decide manually whether to adopt the generated code fragment and

integrate it into the existing code.

The proposed fault prevention method is evaluated in the experiments that transform SOFL formal specifications to Java programs. Besides, I choose to compare it to the existing Yu's method with industrial application, which develops a coding fault prevention guideline that describes examples of actual errors and the corrected code and trains the programmers before coding. The experiment results demonstrate a reduction in both requirements-related faults and development time.

3. An automated method for formal-specification-to-code trace linking.

To efficiently support conformance verification and program maintenance, I propose a novel method that automatically builds effective trace links between the components in formal specifications and their corresponding ones in the code. The proposed trace links recovery approach takes the multi-dimensional attributes of specification components into account to supporting a systematic trace links recovery. The principle of the proposed method is established on the common practice in specification-based implementation that the name and structure of a component and its relationships with others are often preserved in the implemented program. The proposed method is comprised of the following steps:

(1) identifying the components in the formal specification and the components in the code, respectively,

(2) extracting the designed multidimensional attributes for each identified specification component and for each identified code component, respectively,

(3) calculating similarity scores between all possible pairs of components formed from the formal specification components and the code components,

(4) ranking the similarity scores to predict accurate trace links.

Using a VDM-SL formal specification of an *AccountSys* and a SOFL formal specification of an ATM system with their corresponding Java implementation as the experiment data, respectively, I have conducted two experiments to evaluate the proposed formal-specification-to-code trace links establishment approach. I also compare it with manual trace links establishment and two commonly used traceability links methods, which are latent semantic indexing (LSI) and vector space model (VSM-cosine). The experiment results show that my approach achieves much higher accuracy than automatic LSI and VSM-cosine methods while it is slightly less effective than the manual approach.

4. A new method for code inspection.

For program verification, I put forward a new code inspection method called Formal Specification Component Attributes-Based Code Inspection (FSCABCI). The method is motivated by the fact that each component in a formal specification has its own attributes concerned with its structure. Despite some variants referring to syntactical changes during the transformation from specifications to programs, the attributes of

each component in a formal specification are semantically maintained in their implementation. This is the principle that supports the proposed inspection method. Thus, the essence of the proposed method is to use inspection to check whether each attribute of each component in the specification is correctly implemented by its corresponding component in code. To provide inspectors with guidance on detecting the discrepancies between the code and the requirements, I derive some implementation reminders based on the attributes. The implementation reminders are some prompts for one of their correct implementations. Our method can provide the inspector with both the specification component attributes to facilitate their verification and the derived implementation reminders to help the inspector understand the code under inspection to some extent.

The FSCABCI method is comprised of the following activities:

- identifying the components in the formal specification,
- recommending the order of the specification components used in inspection,
- extracting the attributes for the specification components,
- deriving the implementation reminders based on attributes for specification components,
- identifying the components in the code,
- linking the specification components to their corresponding code components,
- analyzing the code components against their corresponding specification components with the extracted attributes and derived reminders to detect defects,
- producing an inspection report.

A controlled experiment is conducted to evaluate its performance by comparing it to the existing *formal specification-based inspection* (FSBI) method, which checks whether functional scenarios in the specification are correctly implemented in the program. The result demonstrates that the inspectors using the proposed method find more (inserted) requirements-related faults than others using FSBI.