Automatic Code Verification by Formal Analysis

Szilvási, Krisztián
krisztian.szilvasi.3@gmail.com

Fenyvesi, Róbert
fenyvesr@gmail.com

November 30, 2019

Abstract

Product Development is driven by stakeholder requirements. The larger the developed system, the harder it is to analyze and verify it. Software Projects are no exceptions. This article aims to show how the verification of huge software projects can be performed automatically against the given requirements. Our contribution consists of two parts. Firstly, an artificial neural network, with a set of successful formal descriptions received from human-readable version of requirements and their formal description to learn the models of correctly implemented requirements. Second, a verifier which analyzes the formal descriptions of the requirements and the semantics obtained by semantically analyzing the source code. The proposed solution checks if written code meets the requirements or not. In the end it can make a suggestion how to fix a code.

Keywords — Formal Verification, Formal Requirement Analysis

Contents

1 Introduction

Product Development is driven by stakeholder requirements. The larger the developed system, the harder it is to analyze and verify it. Software Projects are no exceptions. This project aims to show how the verification of huge software projects can be performed automatically against the given requirements. The project spreads across multiple areas of main stream research.

Natural Language Processing (NLP) is used for formalizing the Software Requirement Specification (SRS). Since, natural language is widely understood by stakeholders, it is used as a common way for representing requirements. Representing requirements in natural language suffers from potential problems like ambiguity, inconsistency and incompleteness. A systematic literature review in the last two decades from 1995 till 2016 shows that collecting ambiguous requirements is one of the highest critical challenges in software engineering [?]. Since the advent of software engineering, researchers used formal and semi-formal methods to overcome this problem. However, even when formal and semi-formal languages are used, there is no escape from natural language as the initial requirements are written in natural language [?]. The consequences of ambiguous requirements will lead to excessive efforts, high cost and failure in some software projects. For example, software developers might decide a subjective interpretation of requirements based on their point of view. Ferrari et al. (2014) argued that this subjective interpretation leads to designing software in a different way from what was intended in the requirements [?]. For several decades, SRS processing and analysis has been the focus of research in software engineering discipline. Since natural language is ambiguous, a computer cannot provide full support to analyze SRS in an automatic fashion. Consequently, the analysis of SRS is conducted manually which consumes time, effort and cost. Most importantly, the manual analysis of requirements results in inefficiency and imprecise results [?]. The problem will be more obvious and critical when software projects involve thousands of requirements and hundreds of SRS documents. Conducting verification of thousands of requirements via humans will become extremely expensive [?]. Generally, the primary source of problems in requirement engineering is reliance on humans extensively [?]. This discussion leads to the importance of finding an automatic way for processing SRS. NLP was used as a possible solution to resolve ambiguity and to provide valuable information to the intended software developers. Ryan (1993) argued that: "It is highly questionable that the resulting system from NLP would be of great use in requirements engineering" [?]. Nazir et al. (2017) conducted a systematic literature review on NLP applications for software requirement engineering, he concluded that: "Manual operations are still required on initial plain text of software requirements before applying the desired NLP techniques" [?].

On the other hand the formal semantics of the source code is needed. A formal semantics should serve as a solid foundation for any programming language development, so it must be correct and complete (to be trusted and useful), executable (to yield a reference implementation), and appropriate for program reasoning and verification.

The five most popular programming languages according to GitHub in 2019 is JavaScript, Python, Java, PHP and C#. Several efforts to give JavaScript a formal semantics have been made, most notably by Politz et al. [?] and Bodin et al. [?]. But Unfortunately, these address fragments of the language and are not fully validated with a formal JavaScript semantics. Having to define two or more different semantics for a real-life language, together with proofs of equivalence, is a huge burden in itself, not to mention that these all need to be maintained as the language evolves. Due to the functional nature of their interpreters, these semantics cannot handle the nondeterminism of JavaScript well. Finally, their interpreters are not suited for symbolic execution, and thus for developing program reasoning tools.

References

Application Programming Interface (API)
API

2 Methods

3 Results

4 Discussion

Acronyms

API Application Programming Interface. 3, *Glossary:* API

NLP Natural Language Processing. 3, *Glossary:* NLP

SRS Software Requirement Specification. 3, *Glossary:* SRS

Glossary

API An Application Programming Interface (API) is a particular set of rules and specifications that a software program can follow to access and make use of the services and resources provided by another particular software program that implements that API . 3

NLP Natural language processing (NLP) is a subfield of linguistics, computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyze large amounts of natural language data. . 3

SRS A software requirements specification (SRS) is a description of a software system to be developed. It is modeled after business requirements specification (CONOPS), also known as a stakeholder requirements specification (StRS). The software requirements specification lays out functional and non-functional requirements, and it may include a set of use cases that describe user interactions that the software must provide to the user for perfect interaction. Software requirements specification establishes the basis for an agreement between customers and contractors or suppliers on how the software product should function (in a market-driven project, these roles may be played by the marketing and development divisions). Software requirements specification is a rigorous assessment of requirements before the more specific system design stages, and its goal is to reduce later redesign. It should also provide a realistic basis for estimating product costs, risks, and schedules. Used appropriately, software requirements specifications can help prevent software project failure. The software requirements specification document lists sufficient and necessary requirements for the project development. To derive the requirements, the developer needs to have clear and thorough understanding of the products under development. This is achieved through detailed and continuous communications with the project team and customer throughout the software development process. The SRS may be one of a contract's deliverable data item descriptions or have other forms of organizationally-mandated content. . 3