WYDZIAŁ INFORMATYKI
Politechniki Białostockiej
15-351 Białystok, ul. Wiejska 45A
tel. +48 85 746 90 50, fax +48 85 746 90 57
e-mail: wi.sekretariat@pb.edu.pl
www.wi.pb.edu.pl

Białystok 1 grudnia 2023

Prof. Ndzw. Stanisław Jarząbek

Politechnika Białostocka

Recenzja Rozprawy Doktorskiej mgr inż. Wiktora Nowakowskiego pt. : „Semantyka translacyjna dla języka specyfikacji wymagań"

Przedmiot, cel i zakres rozprawy

Autor podjął temat automatycznej generacji kodu z wyższego poziomu specyfikacji programów jako środka do zwiększenia produktywności programistów. Warto zauważyć że temat ten od wielu dekad podejmowany jest w badaniach naukowych a także w rozwiązaniach przemysłowych z uwagi na potencjał jaki kryje się w zastąpieniu manualnego kodowania programów przez automatyczną generację. Praca kandydata jest zatem w ważnym nurcie badań który nadal oferuje, oprócz możliwych usprawnień praktyki programowania, także trudne wyzwania.

Za cel teoretyczny pracy autor postawił zdefiniowanie semantyki translacyjnej modeli opisujących wymagania programów, na tyle precyzyjnej aby na jej podstawie można było generować kod. Cel ten autor osiągnął definiując semantykę translacyjną dla języka specyfikacji wymagań RSL.

W części praktycznej autor zaimplementował zaproponowane reguły translacyjne w środowisku ReDSeeDS, z wykorzystaniem języka MOLA. Aby zweryfikować swoje rozwiązanie, autor zastosował je do przykładowej aplikacji.

Zaproponowane metody są oryginalnego pomysłu autora i prowadzą do ważnych usprawnień praktyki programowania.

Analiza istniejącego stanu wiedzy i teza pracy

Proponowane rozwiązania często koncentrują się na generacji niektórych aspektów programu, na przykład szkieletu implementacji klas albo interfejsów użytkownika z koncepcyjnych modeli klasowej struktury programu. Bardziej zaawansowane rozwiązania, rozwijane najczęściej w ramach badań naukowych, bazują na modelach programów (np. UML) wzbogaconych o informacje semantyczną. W wielu dziedzinach aplikacyjnych powstały wyspecjalizowane języki (Domain-Specific Languages, DSL) pozwalające na tworzenie generatorów programów w danej dziedzinie aplikacyjnej. W odróżnieniu od nurtu badań DSL, autor postawił sobie ambitny cel rozwiązania problemu na gruncie ogólnym, w sposób niezależny od dziedziny aplikacji, i orientując się na generacje kompletnego kodu aplikacji z wyższego poziomu specyfikacji programu. Brak jest na dzień dzisiejszy efektywnych rozwiązań ogólnych, w postaci języków specyfikacji pozwalających na adekwatne odzwierciedlenie wszystkich informacji potrzebnych do generacji pełnego kodu warstwy logiki dziedzinowej z poziomu specyfikacji wymagań.

W ciągu ostatnich dekad, miało miejsce szereg prób znalezienia ogólnej metody generacji kodu w języku imperatywnym z nie-operacyjnych specyfikacji programów. Pierwszą dużą próba był projekt Automatic Programming na Uniwersytecie Berkeley w latach 1980-ych. Projekt ten nie zakończył się sukcesem. Analizę swoich doświadczeń i przyczyn niepowodzenia autorzy udokumentowali w artykule *A 15 Year Perspective on Automatic Programming* (IEEE TSE, 1985).

Następna próba miała miejsce w czasie popularności narzędzi CASE (Computer-Aided Software Engineering). Tu ideą była generacja kodu programu z opisu wymagań w postaci diagramów - *Software Through Pictures*. I ta próba również nie przyniosła spodziewanych wyników. Problem nieustannie dotyczył trudności generowania logiki biznesowej programu.

Po ustabilizowaniu języka modelowania UML, w IBM miała miejsce następna próba rozwiązania problemu dekorując modele UML specyfikacjami semantyki programu. Rozwiązanie możliwe w teorii, okazało się nieosiągalne w praktyce z powodu komplikacji połączeń pomiedzy modelami UML wzbogaconymi o semantykę.

Autor omówił podejście Model-Driven Development w jego wielu konkretnych formach, a także inne metody generacji kodu we Wprowadzeniu i w Rozdziale 2.

Istota i jakość rozwiązań zaproponowanych w pracy

Autor oparł swoje rozwiązanie na środowisku ReDSeeDS (Requirements-Driven Software Development System) obejmującym języki, narzędzia oraz proces, wspólnie umożliwiające tworzenie modeli wymagań, architektury i projektu, oraz transformacji przekształcających modele programowe z wyższego poziomu na kod programu. Ważnym elementem ReDSeeDS jest Requirements Specification Language (RSL) - język specyfikacji wymagań ogólnego przeznaczenia (w odróżnieniu od języków DSL).

Oryginalnym wkładem autora jest definicja semantyki translacyjnej dla języka RSL, jako zestaw reguł przekształcania konstrukcji gramatycznych RSL na konstrukcje języka Java. W trakcie tego przekształcenia, struktury RSL są zamieniane na struktury Java - zmienia się składnia abstrakcyjna przy zachowaniu semantyki. Przekształcenia wykorzystują UML, przy czym fragmenty kodu aplikacyjnego dekorują elementy modelu UML. Kod aplikacyjny jest zatem bardziej syntetyzowany niż generowany.

Metoda w sposób poprawny i kompletny prowadzi od specyfikacji programu w RSL do kodu w języku Java. Opis proponowanej metody generacji jest również poprawny i kompletny (Rozdział 5), choć nie łatwy w czytaniu ze względu na detaliczny poziom. Sugestie poprawy czytelności umieściłem w sekcji *Uwagi polemiczne*.

Praca napisana jest dobrą polszczyzną i starannie edytowana, niemniej jednak uwzględnienie sugestii przedstawionych w sekcji *Uwagi polemiczne* mogłoby zwiększyć jej czytelność.

Eksperymentalna ewaluacja zaproponowanych rozwiązań

Autor uzasadnił proponowane metody na gruncie teorii w sposób jasny i przekonywujący. Ostatecznym testem dla metod inżynierskich jest zawsze praktyka.

Rozdział 7 opisuje zastosowanie proponowanej metody do tworzenia przykładowej aplikacji Pet Clinic, często używanej do demonstracji metod tworzenia aplikacji internetowych. Przykład ten pokazuje możliwość zbudowania kompletnej aplikacji przy użyciu proponowanej metody. Praktyczną stronę stosowania metody autor potwierdził w eksperymencie z udziałem studentów. Dalszej eksperymentacji potrzeba dla wszechstronnej ewaluacji korzyści

inżynierskich zastosowania proponowanej metody, i dla zaobserwowania obszarów ewentualnych usprawnień, na przykład w formie stosownych narzędzi.

Uwagi polemiczne i dyskusyjne

1. Czytając pracę niejednokrotnie miałem problem z odróżnieniem oryginalnego wkładu kandydata od wkładu jego współpracowników bądź współautorów publikacji. Warto aby kandydat precyzyjniej określił swój wkład w opisywane rozwiązania, bo w publikacjach często widnieje długa lista autorów.
2. Nie zupełnie zgadzam się z opinia wyrażona we Wprowadzeniu: „Inżynieria wymagań, w małym stopniu koncentruje się na tworzeniu rozwiązań, które prowadziłyby do zmiany paradygmatu [14] [15], np. rozwoju notacji, metod i narzędzi pozwalających formułować wymagania w sposób umożliwiający ich automatyczne przekształcanie w kod.". Generacja kodu z wyższego poziomu specyfikacji programów od wczesnych lat 80-ych ubiegłego wieku przyciąga dużo uwagi zarówno w badaniach akademickich jak i w ramach prac nad narzędziami wspomagającymi tworzenie oprogramowania. Niektóre z tych trendów autor omówił w Rozdziałach 1 i 2, niektóre inne wspominałem w tej recenzji. Sugeruje żeby przeformułować wyżej wspomniane stwierdzenie.
3. W obecnej formie Wprowadzenie (Rozdz. 1) jest zbyt ogólnikowe. Niewiele mówi o specyfice podejścia proponowanym przez autora i nie pozwala zobaczyć na czym polega jego nowatorstwo. Pomocne byłoby aby już we Wprowadzeniu autor rozróżnił kategorie specyfikacji programów będących przedmiotem generacji kodu, i umiejscowił proponowane w pracy podejście w odniesieniu do tych kategorii. Przykładowo:
    a. Specyfikacje programów w obrębie danej dziedziny zastosowań (DSL) vs. specyfikacje programów nie ograniczające się do konkretnej dziedziny,
    b. Opis strukturalnych aspektów programów vs. specyfikacja logiki biznesowej (algorytmiczny aspekt)
    c. Specyfikacja pozwalająca generować kompletny program vs. specyfikacja pozwalająca generować szkielet programu który musi być ręcznie uzupełniony przez programistów.
4. Pożądane byłoby bardziej szczegółowe omówienie oryginalnych rozszerzeń autora względem wcześniejszych prac w ReDSeeDS (jest to wspomniane na końcu Rozdziału 2.6).
5. Jaki jest związek pomiędzy metodami zwinnymi (Rozdział 2.3) a podejściem autora, lub generalnie – metodami generacyjnymi?
6. Czy metody translacyjne proponowane przez autora obejmują zarówno funkcjonalność programu opisaną wymaganiami jak i specyfikę platformy (MDA Rozdział 2.2)?
7. Proszę wyjaśnić wkład autora w kontekście istniejącego środowiska ReDSeeDS przedstawionym na Rysunku 5. Jak rozumiem praca autora koncentruje się na regułach translacyjnych – czy zastany przez autora ReDSeeDS posiadał już reguły translacyjne które trzeba było udoskonalić czy też reguły translacyjne są w całości wkładem autora? Czy autor opracował koncepcje tych reguł czy też skupił się na ich implementacji?
8. Pojęcie Software Case jest użyte ale nie jest wyjaśnione (str 24 i 25), warto może umieścić Rysunek 6 wcześniej. Jaki jest wkład autora pracy w definicje języka SCL (opisującego cały proces generacji, Rysunek 6)?
9. Jaki jest wkład autora w definicje języka RCL opisanego w Rozdział 3 str 34-71, i jego meta-modelu Rozdział 4 str. 72-97?
10. Opis środowiska translacyjnego RSL (Rozdz. 5.1) jest zorientowany na implementacje. Oczywiście taki detaliczny opis jest potrzebny, ale proponowane przez autora rozwiązanie byłoby lepiej zrozumiałe dla czytelnika gdyby zacząć od opisu na poziomie koncepcyjnym, nie implementacyjnym. Wyjaśnić rozmaite rodzaje reguł translacyjnych z jakimi mamy do czynienia i poprzeć opis przykładami konkretnych reguł. Przykłady takich reguł znajdują się w następnym rozdziale w odniesieniu do elementów implementacyjnych i nie są łatwe

w czytaniu. Z uwagi na charakter rozwiązania zdaje sobie sprawę że może nie być łatwo przedstawić rozwiązanie na poziomie koncepcyjnym jakiego mi tu brakuje.

11. W końcowej części pracy warto by dokonać szczegółowej analizy porównawczej podejścia proponowanego przez autora i innych podejść do problemu generacji kodu z wyższego poziomu specyfikacji, omówionych w początkowych rozdziałach pracy.

Podsumowanie

Rozprawę charakteryzuje nowatorstwo rozwiązań, potencjał praktycznych zastosowań jak również możliwość dalszych badań nad generacja programów. Materiał jest dobrze zorganizowany i sama rozprawa napisana w sposób przystępny, z wystarczająca dozą detali i przykładów umożliwiających zrozumienie przedstawianych metod i ich krytyczną ocenę.

Biorąc powyższe pod uwagę oraz uwzględniając wymagania zdefiniowane przez odpowiednią Ustawę o stopniach i tytułach naukowych, stwierdzam, że moja ocena rozprawy jest zdecydowanie pozytywna i proponuję dopuszczenie magistra Wiktora Nowakowskiego do dalszych etapów przewodu doktorskiego.

X Stanisław Jarząbek

Stanislaw Jarzabek